uc3m | Universidad Carlos III de Madrid

Departamento de Ingeniería de Sistemas y Automática

TRABAJO FIN DE GRADO

MÓDULO DE RECONOCIMIENTO GESTUAL PARA CONTROL DE ROBOT EN TAREAS DE ASISTENCIA II

Autor: Pedro Lázaro Sánchez

Tutor: Edwin Daniel Oña Simbaña

Leganés, Septiembre 2017

Resumen

Este trabajo final de grado (TFG) va a tratar el desarrollo de un sistema control sin contacto para el brazo robótico poliarticulado AMOR de 7 grados de libertad. El control se hará por reconocimiento gestual utilizando un dispositivo Leap Motion que es capaz de monitorizar las manos y los antebrazos. El sistema control va a estar basado en la plataforma Robotic Operative System (ROS) y se comunicará con el programa de control de la API de AMOR que está basada en Yet Another Robot Platform (YARP).

Este TFG está orientado hacia el ámbito de la robótica asistencial para incrementar la independencia de personas discapacitadas. Se han realizado diferentes ejercicios simulando tareas cotidianas como la apertura y cierre de un microondas o el agarre de una botella de agua.

Palabras clave: brazo robótico, Leap Motion, ROS, comunicación ROS-YARP, robótica asistencial, reconocimiento gestual

Abstract

This project assesses the development of a non-contact control system for an AMOR polyarticulated robotic arm with 7 degrees of freedom. A Leap Motion gesture recognition system which is capable of monitoring hands and forearms will be used to control the robotic arm. The control system will be based on a Robotic Operative System (ROS) platform and will communicate with the AMOR's API control program, which is based on Yet Another Robot Platform (YARP).

This project is aimed towards the sphere of assistive robotics to improve the independence of people with disabilities. Everyday tasks, such as the opening and closing of the door of a microwave or gripping a water bottle, have been carried out.

Keywords: Robotic arm, Leap Motion, ROS, ROS-YARP communication, assistive robotics, gesture recognition

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Índice

1.1. Motivación

Hoy en día la sociedad en la que vivimos ha experimentado multitud de avances científicos en las últimas décadas que han aumentado considerablemente la esperanza de vida y ha provocado que el porcentaje de ancianos se esté incrementando progresivamente. Según la Organización Mundial de la Salud [1] en 2014, la esperanza de vida se ha incrementado entre 1990 y 2014 unos 6 años de media y, en los países de desarrollados actualmente se encuentra rondando los 76 años. Además España tiene el segundo puesto en la esperanza de vida de las mujeres, 85.1 años, sólo por detrás de Japón.

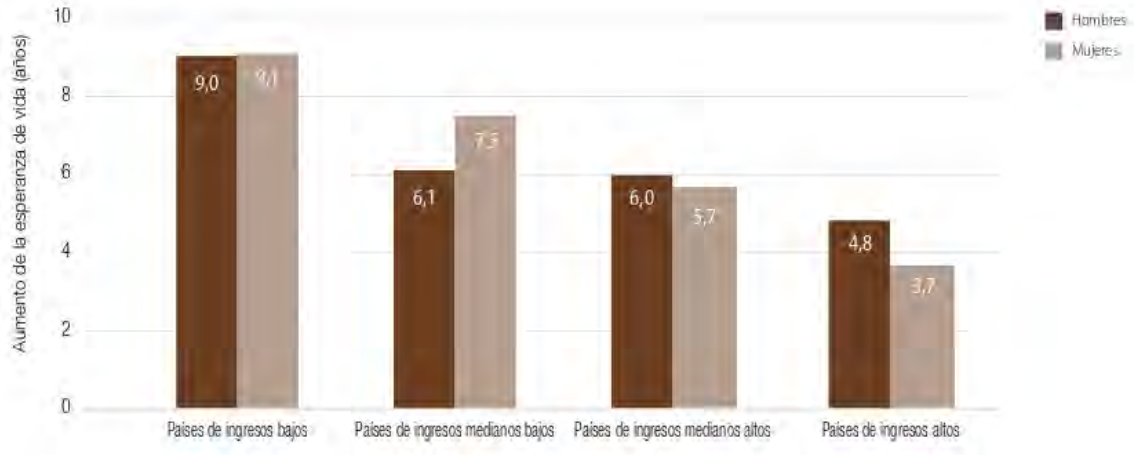


Figura 1.1: Años de esperanza de vida ganados entre 1990 y 2012

Estas personas necesitan cada vez más y más asistencia médica en su día a día porque suelen tener problemas de movilidad, como con enfermedades como el parkinson o debido al propio envejecimiento. Ésto reduce drásticamente la independencia de estos individuos alimentado también una reducción de las relaciones personales, ya sea por pérdida de familiares y amigos o por otros motivos, que promueve enfermedades como la depresión.

Así que la robótica asistencial contribuye a mitigar estos problemas para aumentar la independencia de personas con discapacidad para realizar actividades básicas de la vida diaria (ABVD). En esta línea, una de las aplicaciones más relevantes es el uso de robots de asistencia teleoperados. Debido a las diferentes limitaciones motoras de los usuarios objetivo, facilitar el control de estos robots es de suma importancia.

1.2. Objetivos y Contenido

En este trabajo fin de grado (TFG) va a abordar el control del movimiento del brazo robótico poliarticulado AMOR de 7 grados de libertad mediante un dispositivo llamado Leap Motion que es capaz de monitorizar sin contacto alguno las manos y todos sus dedos. Además este sistema se va a desarrollar bajo la plataforma de software llamada Robotic Operative System (ROS) que facilita una programación modular. Se compilará y se ejecutará en un portátil MSI-CX61 2QC.

El objetivo principal del proyecto es diseñar un modo de control fácil y sencillo. Es imperativo que el sistema sea intuitivo para aumentar su usabilidad y poder abarcar el mayor numero de usuarios, estén o no familiarizados con esta tecnología. Una de las claves del manejo es el uso de gestos lo más sencillos e intuitivos posibles, como por ejemplo, que al cerrar la mano se cierre la garra, sin embargo esto significa que el sistema requiere una movilidad mínima para poder realizarlos.

En cuanto al control cinemático del robot se realizará en función de la ubicación de las manos y se estudiará el uso de varios métodos control para incrementar al máximo el rango de acción del robot.

La estrucutra de la memoria del proyecto es la siguiente:

El **capítulo 2** se centrará en hacer una introducción sobre el área en el que se encuentra este trabajo, en este caso la robótica haciendo incapié en la robótica asistencial. A continuación, se expondrá una visión general de ROS y los conceptos más importantes que se van a usar en este trabajo. En cuanto al hardware, se realizará una introdución de las características de Leap Motion, cómo funciona y qué tipo de información recoge, y una explicación de la estructura del Robot AMOR y su funcionamiento. Por último, se hará un breve resumen de otros trabajos relacionados para conocer cómo han sido resueltos y los planteamientos e ideas que se han utilizado.

En el **capítulo 3** se abordará el desarrollo del TFG. Se comenzará con una visión

general del proyecto y el planteamiento de las hipótesis iniciales que se plantearon a la hora de realizar el proyecto. A continuación se expondrán las distintas fases del proyecto explicando en qué consistieron relacionándolas con las hipótesis planteadas, tanto las que se aplicaron como las que se descartaron. Por último se hablará detalladamente sobre la solución final, su funcionamiento y todos los elementos introducidos junto con la justificación de su uso.

El **capítulo 4** se encuentra el análisis de los resultados obtenidos donde se comentararán las pruebas realizadas y se realizará una crítica sobre los puntos fuertes y débiles del proyecto.

Por último, en el **capítulo 5**, se realizará una conclusión final sobre el proyecto, una exposición sobre algunas propuestas de mejora y posibles ampliaciones, un estudio socio económico y el presupuesto del mismo.

Capítulo 2

Antecedentes y Estado del Arte

Índice

2.1. Robótica

A día de hoy, la robótica es un área multidisciplinar donde confluyen bastantes campos de estudio diferentes, desde la informática y la electrónica hasta la biología y la medicina pasando por la mecánica, física , matemáticas o las tecnologías de la comunicación.

En cuanto a su definición todavía no es realmente concisa,por ejemplo, según la RAE [2] la **robótica** *es una técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales* aunque es muy común encontrarse con la definición de que *es la ciencia o rama de la tecnología, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren del uso de cierta inteligencia.* Si bien es cierto que ambas definiciones son muy parecidas, se pueden apreciar matices como que en el caso de la RAE centra su significado en la sustitución del ser humano por parte del robot mientras que en la segunda definición no resulta imprescindible.

En realidad, la razón por la que el significado de robótica esté tan difuso se debe a que todavía no hay ningún consenso sobre el significado de robot. Dado que el uso más importante de los robots aparece en el ámbito industrial, las primeras definiciones encontradas están referidas a dicho campo.

Una de las primeras definiciones fue dada por la Asociación de Robótica Industrial (RIA) en 1979 y considera que un **robot industrial** *es un manipulador multifuncional y reprogramable, diseñado para mover materiales, piezas, herramientas o dispositivos especiales, mediante movimientos programados y variables que permiten llevar a cabo diversas tareas.*

Por otro lado, según la Asociación Internacional de Estándares (ISO) en la norma ISO 8373 un **robot industrial** es *un manipulador automáticamente controlado, reprogramable, multifuncional, programable en tres o más ejes, que puede estar fijado en un lugar concreto o ser móvil, para su uso en aplicaciones industriales.*

Según la Asociación Japonesa de Robótica (JARA), se considera como **robot** a *cualquier dispositivo capaz de moverse de modo flexible análogo al que poseen los organismos vivos, con o sin funciones intelectuales, permitiendo operaciones en respuesta a órdenes humanas.*

Por último, según la RAE un **robot** es *una máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a las personas.*

Realmente es difícil definir qué es un robot porque, simplemente, cada vez hay más tipos de robots distintos y cualquiera de las definiciones enunciadas anteriormente termina excluyendo alguna clase así que, quizás se podría afirmar que una de las definiciones más acertadas es la que dio **Joseph Engelberger**, unos de los grandes pioneros de la Robótica y uno de los desarrolladores del primer robot industrial, que dijo *"No puedo definir lo que es un robot, pero reconozco uno cuando lo veo"*.

Así que, como se puede observar que el significado de robot es algo difuso, primero se va explicar cuál es el origen de la palabra *robot* y se expondrá un breve resumen de la historia de la Robótica.

2.1.1. Historia de la Robótica

La palabra *robot* procede del checo *robota* que significa servidumbre o trabajador forzado. Este término se empleó por primera vez en la obra de teatro llamada *Rossum's Universal Robots* publicada por Karel Čapek que trata sobre la historia de un científico brillante, llamado Rossum, y su hijo que se dedican a fabricar robots para que sirvan al ser humano. Al final de la obra los robots se rebelan contra sus dueños, destruyendo a la humanidad.

Este término fue recogido y popularizado por la ciencia ficción con escritores como Isaac Asimov que en sus libros se imaginó las 3 leyes famosas de la robótica:

- **Primera ley:** "Un robot no le hará daño a un ser humano o, por inacción, permitir que un ser humano sufra daño"
- **Segunda ley:** "Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entran en conflicto con la primera ley"
- **Tercera ley:** "Un robot debe proteger su propia existencia en la medida que su protección no entre en conflicto con la primera y segunda ley"

En cuanto a la historia de la Robótica, se suele considerar que comenzó en 1960 con el desarrollo del primer **Robot Industrial** por parte de **George Devol** y **Joseph Engelberger** pero, desde la antigüedad, el ser humano ha construido máquinas que podrían definirse como robots. Por ejemplo, los egipcios y los griegos diseñaron estatuas con sistemas mecánicos e hidraúlicos para fascinar a los fieles, o, durante los siglos XVII y XVIII, se construyeron muñecos mecánicos muy ingeniosos.

Con la llegada de la revolución industrial se diseñaron grandes invenciones mecánicas como la hiladora giratoria de Hargreaves (1770), la hiladora mecánica de Crompton (1779), el telar mecánico de Cartwright (1785), el telar de Jacquard (1801), y otros que bien podrían ser los antecesores más próximos de los robots industriales.

A continuación, en la tabla 2.1, se pueden observar algunos de los hitos más importantes hasta 2011 según la Federación Internacional de Robótica (IFR) en su página web [3] aunque hay que destacar que principalmente está enfocado desde el punto de vista de la robótica industrial.

Tabla 2.1: Cronología del desarrollo de los Robots en los últimos años

Año	**Descripción**
1959	Desarrollo del primer robot industrial
1961	Instalación del primer robot industrial
1962	Primer robot cilíndrico
1967	Primer robot industrial en Europa
1968	Marvin Minsky desarrolla el brazo robótico tentáculo
1969	Se desarrolla la visión robot
1969	Primer robot de pintura
1971	Primera linea de producción en Europa
1971	Se crea la Asociación Japonesa de Robótica (JIRA)
1973	Primer robot con 6 ejes de libertad
1974	Primer robot eléctrico industrial controlado por microprocesador
1975	Primer robot cartesiano utilizado en ensamblaje
1975	ABB desarrolla un robot que soporta 60 kg
1976	Robots en el espacio
1978	PUMA, desarrollado por Unimation
1978	Hiroshi Makito desarrolla el SCARA-Robot
1980	Primer uso de un robot con visión
1982	IBM desarrolla el lenguaje AML
1984	ABB produce el IRB 1000
1992	Wittmann introduce el control CAN-Bus para robots
1992	ABB lanza un sistema de control abierto
1994	Motoman introduce el primer sistema robótico de control sincronizando 2 robots
1996	KUKA lanza el primer sistema robótico de control basado en PC
1999	Primer diagnóstico remoto a través de internet por KUKA
2003	Los robots van a Marte
2006	Comau introduce el primer Wireless Teach Pendant
2006	KUKA presenta el primer robot de bajo peso
2007	KUKA lanza el primer robot que soporta 1000 kg
2008	FANUC lanza un nuevo robot que soporta 1200 kg
2009	Motoman introduce un sistema de control capaz de sincronizar 8 robots
2011	El primer robot humanoide en el espacio

2.1.2. Clasificación de la robótica

En la actualidad, el campo de la robótica está en pleno auge lo que ha promovido el desarrollo de múltiples tipos de robots diferentes e impide que haya un consenso en cuanto a cuáles son los tipos de robot que existen. No obstante, han aparecido bastantes formas de clasificación diferentes como según la arquitectura, según la generación, según el nivel de control, según el tipo de control o según la aplicación.

2.1.2.1. Según la arquitectura

La primera de todas es según la arquitectura que clasifica a los robot en función de la apariencia física y visual que tienen. Podemos encontrar cinco tipos distintos: Poliarticulados, Móviles, Androides, Zoomórficos e Híbridos.

Poliarticulados

En este grupo se encuentran robots de muchas formas y configuraciones cuya característica común es que se tratan de robot básicamente sedentarios, aunque excepcionalmente pueden ser guiados para realizar desplazamientos limitados y, además, están estructurados para mover sus elementos terminales (pinzas de sujección, herramientas, elementos de soldadura, etc) en un determinado espacio de trabajo según uno o más sistemas de coordenadas y con un número reducido de grados de libertad.

Por lo general se consideran como poliarticulados a manipuladores y algunos robots industriales, como en la figura 2.1, y se suelen utilizar cuando en necesario abarcar una zona de trabajo relativamente amplia o alargada, actuar sobre objetos con un plano de simetría vertical o disminuir el espacio ocupado en la base.

Figura 2.1: Robot poliarticulado

Móviles

Según la normativa ISO un robot móvil es *todo aquel que este montado sobre una base que sea capaz de moverse de forma automática* [4].

A diferencia de los anteriores, estos robots cuentan con una gran capacidad de desplazamiento, y están basados en carros o plataformas con algún tipo de un sistema locomotor rodante aunque también existen robots aéreos o submarinos. Además, se pueden mover siendo controlados de forma telemática o guiándose a partir de la información recibida de su entorno a través de sus sensores pudiendo incluso llegar a sortear obstáculos.

Existen múltiples aplicaciones para este tipo de robots y abarcan prácticamente casi todas las áreas de actuación, por ejemplo, se suelen utilizar de robots domésticos para limpiar el suelo como el de la figura 2.2a pero, por otro lado, también tienen aplicaciones militares como los UAVs (Unmanned Aerial Vehicles), figura 2.2b, pasando por el área industrial para transportar piezas de un punto a otro en una cadena de fabricación o en el área espacial como el Curiosity, figura 2.2c.

(a) robot doméstico

(b) UAV

(c) Curiosity

Figura 2.2: Robots Móviles

Androides

Un Androide es cualquier robot que intenta reproducir total o parcialmente la forma y el comportamiento cinemática del ser humano, figura 2.3. Los androides son muy populares en la ciencia ficción, sin embargo, en la actualidad todavía son dispositivos poco evolucionados sin utilidad práctica, y suelen estar destinados, principalmente, al estudio y experimentación.

Uno de los aspectos más complejos de estos robots, y sobre el que se centra la mayoría de los trabajos, es el de la locomoción bípeda. En este caso, el principal problema es controlar dinámica y coordinadamente en el tiempo real el proceso y mantener simultáneamente el equilibrio del robot.

Figura 2.3: ASIMO

Zoomórficos

Los robot zoomórficos, figura 2.4, se caracterizan principalmente por el uso de sistemas de locomoción que imitan a los diversos seres vivos. Según esta definición se podría aceptar que los androides también entrarían en esta clase de robot, sin embargo, normalmente se suelen relacionar más con animales o insectos. Hay dos tipos distintos: robots zoomórficos caminantes y no caminantes.

Los robots zoomórficos caminantes multípedos, figura 2.4a, son bastante numerosos y se están desarrollando con el objetivo de crear verdaderos vehículos todoterrenos, pilotados o autónomos, capaces de evolucionar en superficies muy accidentadas. Además, tienen aplicaciones muy interesantes como en el campo de la exploración espacial o en el estudio de los volcanes.

En cuanto al grupo de los Robots zoomórficos no caminadores, actualmente no hay tanto nivel de investigación. Se reducen a robots que imitan a gusanos o serpientes asimilando el movimiento de los mismos, figura 2.4b.

(a) Robot Araña

(b) Robot Serpiente

Figura 2.4: Robots Zoomórficos

Híbridos

En esta clase entran los robots de difícil clasificación, cuya estructura se sitúa en combinación con alguna de las anteriores ya expuestas, bien sea por conjunción o por yuxtaposición. Por ejemplo, un dispositivo segmentado articulado y con ruedas, tiene al mismo tiempo atributos de los robots móviles y de los robots zoomórficos. Así mismo pueden considerarse híbridos algunos robots formados por la yuxtaposición de un cuerpo formado por un carro móvil y de un brazo semejante al de los robots industriales. También se encontrarían en esta situación algunos robots antropomorfos como el de la figura 2.5 que no pueden clasificarse ni como móviles ni como androides.

Figura 2.5: Robot Híbrido

2.1.2.2. Según la generación

Esta sección clasifica a los robot en base a su complejidad y los divide en generaciones que están relacionadas con su desarrollo y los avances conseguidos a lo largo de la historia. Este ranking consta de cuatro generaciones consolidadas y una quinta que se corresponde con los avances de la época actual.

- **Primera Generación:** El sistema de control de los robots de esta generación está basado en la "paradas fijas" mecánicamente.Esta estrategia es conocida como control de lazo abierto. En este tipo se encuentran los robots **manipuladores** que son sistemas mecánicos multifuncionales con un sencillo sistema de control, bien manual, de secuencia fija o de secuencia variable. Se suelen denominar como robots Play-back y un ejemplo serían los mecanismos de relojería que permiten mover las cajas musicales o los juguetes de cuerda.

- **Segunda Generación:** En esta generación se encuentran los robots comunmente llamados **de aprendizaje** y . En este caso son más conscientes del entorno debido al uso de sistemas de lazo cerrado que, a través de sensores, obtienen información del entorno y actúan en consecuencia. Además son capaces de aprender y memorizar secuencias de movimientos mediante el seguimiento de los movimientos de un operador humano y de ahí su sobrenombre.

- **Tercera Generación:** A medida que se desarrollaba la generación anterior, se empezaron a necesitar el uso de computadoras que pudiesen soportar lenguajes de programación y así es como surgieron los robots **con control sensorizado**. Estos robots también usan sistemas de control de lazo pero en este caso se da un paso mas y se vuelven reprogramables. Mediante las computadoras son capaces de analizar la información captada de su entorno mediante sensores. Además comienza a desarrollarse la visión artificial.

- **Cuarta Generación:**Esta generación se caracteriza por el uso de sensores mucho mas sofisticados que mandan información al controlador y pueden analizarla mediante estrategias complejas de control. Se suelen denominar como robots **inteligentes** debido a su capacidad de adaptación y aprenden de su entono mediante el uso de "conocimiento difuso" o "redes neuronales" que permiten una mayor capacidad de reacción en tiempo real.

- **Quinta Generación:** Actualmente se encuentra en desarrollo pero están sucediendo grandes avances como la creación de nuevos lenguajes de programación específicamente para robots o el diseño de nuevos modelos de conducta mucho mas complejos. También hay que destacar el desarrollo de áreas como la nanotecnología que seguramente influya enormemente en el futuro de la robótica.

2.1.2.3. Según el nivel de control

En esta clasificación se agrupan los robots en función del grado de complejidad de sus sistemas de control. Se suelen diferenciar entre tres niveles de control: nivel de inteligencia artificial, nivel de modo de control y nivel de servo-sistemas.

1. **Nivel de inteligencia artificial:** Aquí se encuentran los robots que son capaces de identificar un comando y realizar una serie de operaciones de bajo nivel basadas en un modelo estratégico de tareas.

2. **Nivel de modo de control:** En este caso los movimientos del sistema robótico correspondiente son moldeados por lo que llevan incluida la interacción dinámica necesaria para relacionar los diferentes mecanismos, trayectorias planeadas, y los puntos de asignación seleccionados.

3. **Nivel de servo-sistemas:** Es el tipo de robots más simple de los tres donde se utilizan actuadores que controlan los parámetros de los mecanismos mediante el uso de una retroalimentación interna a partir de la información obtenida por los sensores, modificando la ruta prevista si es necesario. Todas las detecciones de fallas y mecanismos de corrección son implementadas en este nivel.

2.1.2.4. Según el tipo de control

Esta forma de clasificación puede parecerse a la anterior pero el enfoque es distinto. En este caso se centra en el modo de control en vez de en su complejidad aunque suelen estar relacionadas. Así que se distinguen 4 tipos (según la norma ISO 8373 y la IFR):

1. **Robot secuencial:** En este caso sólo se pueden controlar una serie de puntos de parada dando lugar a un movimiento punto por punto, como sucede en el caso de algunos manipuladores neumáticos.

2. **Robot controlado por trayectoria:** En este caso ya se puede diseñar la trayectoria que debe de seguir el robot de manera continua, como se hace en los manipuladores industriales.

3. **Robot adaptativo:** Este tipo de robots son capaces de modificar sus tareas de acuerdo a la información recibida del entorno, por ejemplo a través de un sistema de visión.

4. **Robot teleoperado:** Son aquellos que pueden ser controlados remotamente por un operador humano, extendiendo las capacidades sensoriales y motoras de éste a localizaciones remotas.

2.1.3. Aplicaciones

En cuanto a la aplicación práctica de los robots podemos observar dos grupos fundamentales que son los robots industriales y los robots de servicio.

2.1.3.1. Robots industriales

Según la norma ISO 8373 la definición de **robot industral** es un *manipulador automáticamente controlado, reprogramable, multifuncional, programable en tres o más ejes, que puede estar fijado en un lugar concreto o ser móvil, para su uso en aplicaciones industriales*. Se usan para la manufactura de productos, figura 2.6, y hay distintos tipos de robots industriales que realizan diferentes acciones como manipulación (en fundición, moldeado, forja, tratamientos térmicos, etc.), soldadura, pintura, mecanizado, montaje, almacenamiento o control de calidad.

Figura 2.6: Cadena de montaje de vehículos

Por lo tanto son máquinas que están enfocadas a realizar tareas complicadas para el ser humano y reiterativas. Su uso está bastante extendido en las grandes empresas debido a su alta productividad que mejora la competitividad de éstas.

2.1.3.2. Robots de servicio

Según la Federación Internacional de Robótica(IFR), un **robot de servicio** es aquel que *opera de manera semi o totalmente autónoma para realizar servicios útiles a los humanos y equipos, excluyendo las operaciones de manufactura*. Además también aclara que se pueden considerar como de servicio los robots industriales cuando

no estén siendo dedicados a tareas no manufactureras. Pueden estar equipados, o no, con brazos manipuladores como los industriales y a menudo suelen ser móviles.

Este grupo afecta a un gran número de áreas y, a continuación se van a dar unas claves de la influencia de los robot en algunas de ellas y los tipo de robots que se suelen utilizar.

- **Agricultura y Construcción:** En estas áreas se centran en la realización de tareas de fuerza y logística. Es muy común robots manipuladores móviles.
- **Área doméstica:** Estos robots están destinados a ser usados por seres humanos sin una formación específica para ayudarles a sus quehaceres diarios. Suelen estar dedicados a tareas de limpieza por lo tanto suelen ser robots adaptativos móviles con un control sensorizado para conseguir cierta autonomía.
- **Ocio:** Aquí se engloban una gran cantidad de tipos de robots porque existen muchas actividades donde influyen. Se usan tanto en hostelería como en parques temáticos pasando por guías para museos o juguetes. Suelen ser robots bastante automatizados, es decir, robots play-back que ejecuten acciones repetitivas aunque actualmente existen robots juguetes con un cierta inteligencia que son capaces de reaccionar a los eventos de su entorno.
- **Medicina:** En este área donde más ha repercutido el uso de los robot ha sido en la cirugía, debido a su precisión. Por lo tanto son robots que tienen que ser capaces de recoger información del entorno y actuar en consecuencia con la máxima precisión. Suelen ser manipuladores fijos inteligentes que controlan importantes sistemas de sensores.
- **Área asistencial:** Por último se encuentra la robótica asistencial que es el objetivo de este proyecto y que se explicará detalladamente en la próxima sección. Básicamente se centra el intento de otorgar una mayor autonomía a personas con movilidad limitada como, por ejemplo, a través de robots móviles como sillas de ruedas autónomas o manipuladores en forma de prótesis que les faciliten el día a día.

2.1.4. Robótica asistencial

Según la definición de la norma ISO 13482 de 2014 un **robot asistencial** es *aquel que ejecuta acciones que contribuyen directamente en la mejora de la calidad de vida de los humanos, excluyendo las aplicaciones médicas.* Además añade que puede haber contacto entre la persona y el robot para desarrollar las tareas correspondientes y etiqueta como robots asistenciales tres términos: sirviente móvil robot, asistente físico robot y transporte personal robot. Sin embargo no excluye que no pueda haber otros tipos de robots asistenciales que no se encuentren definidos completamente en esas tres opciones.

1. Un **sirviente móvil robot** es *un robot asistencial que es capaz de moverse para realizar tareas interactuando con las personas, como sujetar y agarrar objetos o intercambiar información.*

2. Un **robot de asistencia física** es *un robot asistencial que físicamente ayuda al usuario para realizar las tareas correspondientes otorgando un suplemento o un aumento de las capacidades físicas del usuario.*

3. Un **transporte personal robot** es u*n robot asistencial cuyo propósito es transportar personas a la ubicación correspondiente. Además se añade que puede poseer una cabina, puede estar equipado con algún tipo de asiento o soporte similar y que puede transportar también objetos de la persona correspondiente como mascotas o propiedades.*

A partir de estas definiciones se puede intuir que los robots utilizados deben de tener un cierta autonomía lo que requiere un nivel de inteligencia elevado, estar provistos por gran cantidad de sensores para obtener información del entorno y actuar en consecuencia, y en algunos poseer algún tipo de manipulador. También pueden requerir el poder ser controlados de forma remota.

Respecto a este proyecto se podría considerar que el robot que se va a utilizar se podría considerar como sirviente robot aunque con algunos matices. La definición da a entender, aunque no lo afirma expresamente, que los robots pertenecientes

a dicha clase deben tener cierta autonomía y en este caso el control del robot es completamente teleoperado, es decir, controlado de forma remota. Por otro lado, la palabra móvil puede significar que el robot debería permitir desplazar su base, es decir, debería poder moverse sobre una superficie, sin embargo, en este proyecto el robot mantiene su base fija aunque no se descartaría situarla sobre una superficie móvil ajena al robot.

2.1.4.1. Objetivos

Por lo tanto la robótica asistencial es el área de la robótica que se especializa en el diseño y desarrollo de equipos que interactúan directamente con el individuo para su rehabilitación, bien sea por la perdida de capacidad en la movilidad de sus miembros o bien por la perdida física de uno más de ellos.

Hoy en día, vivimos en una sociedad cada vez más y más envejecida debido, sobre todo, a una gran calidad de vida que ha aumentado considerablemente la esperanza de vida de la población. Esto provoca que cada vez se destinen más recursos y personal al cuidado de nuestros mayores por lo que el desarrollo de la robótica será crucial para facilitar la vida de éstos. Por otro lado, otro de los grandes objetivos es la rehabilitación de pacientes tanto para acelerar la recuperación de los mismos como facilitarles herramientas para mejorar su día a día para paliar sus deficiencias, sean temporales o no, pasando por facilitar los análisis y la recogida de datos para obtener diagnósticos mas precisos. Por último,no hay que olvidar su aplicación como apoyo emocional que puede llegar a ser clave en algunos casos.

Tal y como se ha definido en el punto anterior existen varios tipos de robots dentro de la robótica asistencial. Por ejemplo, existen sillas de ruedas autónomas para pacientes con alguna deficiencia locomotriz importante o se están desarrollando infinidad de prótesis como reemplazo de miembros como brazos, piernas o incluso de sentidos como la vista. También existen exoesqueletos para ayudar al caminar o sistemas robóticos que son capaces de analizar los ejercicios de rehabilitación de un paciente para comprobar si evoluciona correctamente.

2.1.4.2. Ventajas

Las ventajas que pueden ofrecer son enormes simplemente por el uso de robots. A grosso modo el hecho de ser robots les otorga ciertas características contra las que cualquier se humano no puede competir. Primero, ofrecen una disponibilidad de horarios casi ilimitada al no tener necesidades físicas como los humanos, que significa una atención al paciente las 24h del día. Así mismo la precisión y la productividad es envidiable debido sus las altas posibilidades de especialización. Además existen aplicaciones que directamente no podrían ser ejecutadas por un ser humano como es en el caso de las prótesis o exoesqueletos.

Como se ha comentado, en el punto anterior uno de los grandes objetivos se basa en el cuidado de las personas mayores y estudiando las ventajas mencionadas anteriormente. El incremento de ancianos en la sociedad implica una mayor saturación del sistema sanitario lo que provoca cierta desatención a nuestros mayores que influye negativamente en el nivel de vida de éstos por lo que la introducción de la robótica asistencial, indirectamente, mejora la labor del profesional médico al reducir la carga de trabajo de los mismos y mejorando el rendimiento del sistema sanitario. Por otro lado es muy importante destacar la necesidad de apoyo emocional que sufren muchas de las personas mayores debido a la pérdida de contacto con amigos y familiares, ya sea por distancia o tiempo como por otros motivos, por lo que robots especializados en apoyo emocional como NUKA (figura 2.7), cuya eficacia ha sido probada en algunos estudios como en el estudio [5], son necesarios e imprescindibles.

Figura 2.7: Robot NUKA

2.1.4.3. Inconvenientes

En cuanto a los inconvenientes, hay que tener en cuenta la propia naturaleza del ser humano como ser sociable, es decir, la necesidad del contacto social para desarrollarse correctamente. Según el artículo *Personal Care Robots for Older Adultuts: An Overview* [6], el reemplazo del ser humano por parte de los robots puede influir negativamente en el desarrollo social de los individuos. El artículo hace incapié en el caso de la gente de avanzada edad, debido a sus, ya de por sí, escasas oportunidades de relacionarse aunque se podrían extrapolar las conclusiones alcanzadas, prácticamente, para cualquier tipo de paciente.

Como se ha mencionado en el párrafo anterior, la pérdida de contacto con familiares y amigos reduce sus posibilidades sociales por lo que un uso excesivo de robots podría aislar a estos individuos provocando desinterés en temas sociales como la higiene o la imagen. Además, el uso de asistentes personales por parte de la gente mayor disminuye las visitas de familiares, ya que éstos pueden conocer su estado de forma remota, provocando problemas emocionales en estos individuos como la depresión, razón por la cual se ha mencionado antes la importancia del desarrollo de robot especializados para este ámbito.

Por otro lado, a parte de la reducción de las relaciones sociales hay que destacar la poca adaptabilidad que puede tener un robot dado que se basan en unas reglas fijas que otorgan ciertas limitaciones concretas. Éstas pueden generar conflictos con los pacientes al no ser tan flexible como podría ser un médico humano que derivan, por ejemplo, en enfados y, en la actualidad, los robots no están suficientemente desarrollados y prácticamente no tiene ninguna forma de manejar ese tipo de situaciones.

Por último, se debe tener en cuenta que delegar el control de un robot en una persona mayor o de cualquiera edad puede suponer un peligro al no tener, normalmente, conocimientos necesarios sobre su uso y, en el caso de la gente mayor, pueden sufrir algún tipo de episodio que les impida actuar con normalidad.

2.2. ROS

Cada día la comunidad robótica progresa más rápido y esto es debido a que el hardware que necesita cualquier robot es cada vez más accesible y barato. En consecuencia, cada vez se desarrollan algoritmos más complicados que incrementan la autonomía de los robots gracias al uso de gran cantidad de herramientas y en este punto aparece la plataforma **Robot Operating System** (**ROS**) que es un framework desarrollado para ayudar a conectar todas estas herramientas y facilitar la creación de un código para los robots correspondientes.

Según ROS.org [7], *ROS is an open-source, meta-operating system for your robot.* Significa que provee de los servicios que se esperan de un sistema operativo, incluyendo la capa de abstracción de hardware, herramientas de control de bajo nivel, implementación de funcionalidades comunes, comunicación entre procesos y manejo de paquetes pero también provee herramientas y librerias para obtener, construir, escribir y ejecutar código a través de varios ordenadores.

ROS no es realmente un sistema operativo sino un framework con set enorme de herramientas, las cuales proveen la funcionalidad de un sistema operativo en un grupo heterogéneo de computadoras. Su utilidad no se limita a robots pero la mayoría de las herramientas que se proveen se enfocan en trabajar con hardware periférico. ROS posee más de 2000 paquetes, y cada paquete tiene una funcionalidad especializada por lo que el número de herramientas conectadas al framework es uno de sus puntos fuertes. Esto se debe a que ROS está hecho por su propia comunidad por lo que, después de varios años, ha dado lugar a una gran cantidad de paquetes reusables que son fáciles de integrar gracias a la arquitectura del sistema.

La característica clave de ROS es la manera en cómo se ejecuta el software y cómo se comunica, ya que te permite diseñar software complejo sin saber realmente cómo funciona cierto hardware. ROS permite conectar una red de procesos o nodos con un eje central. Así que los nodos se pueden ejecutar desde varios dispositivos y se conectan a ese eje en distintas formas.

Las maneras principales de crear una red son proporcionar servicios que necesiten de una pregunta para enviar la información, o definir conexiones de publicista o suscriptor con otros nodos. Ambos métodos comunican a través de tipos de men-

sajes específicos. Algunos tipos son proporcionados por los paquetes centrales pero los tipos de mensaje pueden ser definidos por paquetes individuales de una forma bastante sencilla y práctica.

Por lo tanto se pueden armar sistemas complejos al conectar soluciones existentes para pequeños problemas. Según este artículo [8], la forma como se implementa el sistema permite:

- Reemplazar componentes con interfaces similares sobre la marcha, quitando la necesidad de detener el sistema para varios cambios.
- Multiplexar salida de múltiples componentes a una entrada para otro componente, permitiendo la solución paralela de varios problemas.
- Conectar componentes hechos en varios idiomas de programación con tan solo implementar los conectores apropiados al sistema de mensajería, haciendo más fácil el desarrollo de software al conectar módulos existentes de varios desarrolladores.
- Crear nodos sobre una red de dispositivos sin preocuparse sobre dónde se ejecuta un código e implementar los sistemas de comunicación entre proceso (IPC) y llamada a procedimiento remoto (RPC).
- Conectar directamente a transmisiones en demanda de hardware remoto sin escribir código extra, esto al emplear los dos puntos anteriores.

Ésto permite que se puedan desarrollar fácilmente soluciones simples e iterativas. Además ROS es multiplataforma por lo que es capaz de conectar procesos de múltiples dispositivos y permite la utilización de cualquier idioma de programación simplemente con determinar o desarrollar las clases correspondientes. Otro punto a favor es el reducido espacio que ocupa, permitiendo integrarlo con otras infraestructuras de software de robots como OpenRAVE.

2.2.1. Conceptos

A continuación se van a exponer y aclarar los conceptos sobre la arquitectura de ROS que se utilizarán a lo largo de este proyecto según la web oficial de ROS [9].

1. **Package:** es la principal unidad de organización en ROS. Un package contiene los procesos que se ejecutan o nodos, una librería dependiente de ROS, los archivos de configuración o cualquier elemento que sea útil organizado de manera compacta. Por lo tanto el package es el elemento mas pequeño e independiente que puede ser compilado.

2. **Nodos:** Son procesos que permiten la comunicación. ROS está diseñado para ser modular y éstos son los elementos imprescindibles para la comunicación. Están definidos en las librerías incluidas en ROS como roscpp o rospy.

3. **Master:** El ROS Master es el núcleo de ROS donde se conectan todos los nodos y se intercambia la información.

4. **Mensaje:** Los nodos se comunican entre sí enviando mensajes. Un mensaje es una estructura comprimida de arrays de datos como integer, floating point, boolean, etc. Además los mensajes pueden incluir arbitrariamente tanto estructuras como arrays.

5. **Topic:** Los mensajes pueden ser enviados a través de un sistema de publicación y subscripción, por ejemplo, un nodo es capaz de enviar información publicándolo en el topic correspondiente. Al mismo tiempo, el nodo que esté interesado en esa información puede suscribirse a ese topic para recibirla. Puede suceder tanto que haya múltiples publishers (nodos publicistas) como subscribers (nodos subscriptores) a un mismo topic o un único nodo que esté conectado a varios topics. Por lo tanto cualquier nodo es capaz de conectarse al topic correspondiente, siempre y cuando utilice el mismo tipo de mensajes.

A continuación, en la figura 2.8 se puede ver un esquema de cómo funciona la comunicación entre nodos donde hay que añadir que para que funcione los nodos implicados deben utilizar el mismo tipo de mensaje. El diseño del esquema utiliza los bloques amarillos para destacar los elementos o programas utilizados, los bloques verdes equivalen a los nodos utilizados y los bloques azules a los topics. Los topic se han situado en el bajo la influencia de ROS Master porque es el elemento donde se conectan los nodos a los topics correspondientes. Además el área gris indica la zona

de influencia de cada elemento o programa

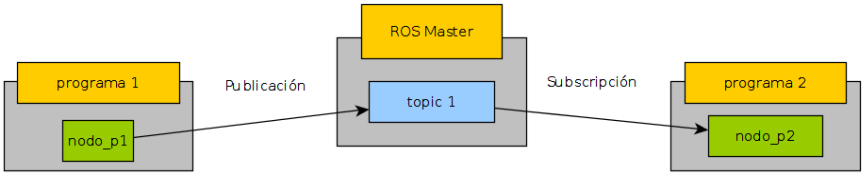


Figura 2.8: Esquema de la comunicación en ROS

ROS también tiene un sistema oficial de compilación llamado **catkin** que combina macros de CMake con scripts de Python para otorgar las funcionalidades de cualquier proceso normal de CMake (para más información *http://wiki.ros.org/catkin*). El flujo de trabajo de catkin es muy similar a CMake pero aporta ciertas herramientas como la búsqueda automática de packages o la compilación de proyectos dependientes y múltiples al mismo tiempo.

El sistema de compilación es el responsable de generar *objetivos* que puedan ser usados luego por un usuario. Estos objetivos pueden ser tanto librerías, programas ejecutables o cualquier cosa que no sea un código estático. En un ROS, cada package consiste en uno o más objetivos una vez compilado. Para compilar, un sistema de compilación necesita información como la localización del código fuente, las dependencias del código y dónde se encuentran, o cuáles y dónde deben ser compilados los objetivos y dónde deben ser instalados. Con CMake se suele utilizar un archivo llamado *CMakeLists.txt* para especificar toda esa información y en el caso de catkin extiende las funcionalidades de CMake para conseguir utilizar dependencias entre packages.

Para ello utiliza dos archivos: package.xml y CMakeLists.txt. El primero es el responsable de la secuencia de configuración y las dependencias de los packages en los espacios de trabajo catkin mientras que CMakeLists.txt es el encargado de preparar y ejecutar la compilación. Para más información aqui se dejan unos enlaces a la wiki de ROS donde explican más detalladamente las funcionalidades de package.xml y CMakeLists.txt respectivamente:

- *http://wiki.ros.org/catkin/package.xml*
- *http://wiki.ros.org/catkin/CMakeLists.txt*

2.3. Leap Motion

Leap Motion es un sistema capaz de reconocer y monitorizar manos, dedos y herramientas similares a un bolígrafo o un lápiz. El dispositivo opera a corta distancia con gran precisión y alta frecuencia de fotogramas por segundo y es capaz de detectar puntos discretos, gestos y movimientos.

Figura 2.9: Leap Motion

2.3.1. Especificaciones técnicas

Las dimensiones de este dispositivo son bastante reducidas en comparación con otros interfaces gestuales del mercado: tan solo mide alrededor de 80 mm de largo, 30 mm de profundidad y 11 mm de alto, figura 2.10.

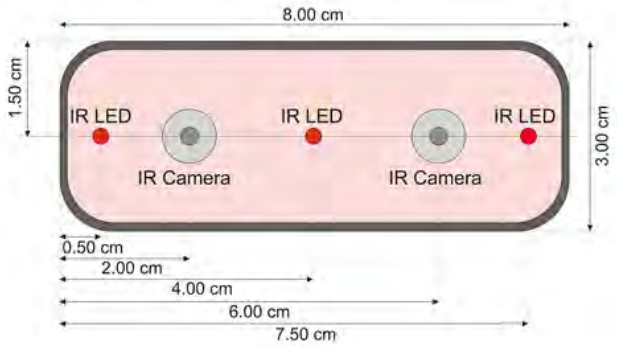


Figura 2.10: Dimensiones de Leap Motion

Y cuenta con 2 cámaras, 3 leds infrarrojos y un microcontrolador, tal y como se puede apreciar en la figura 2.11.

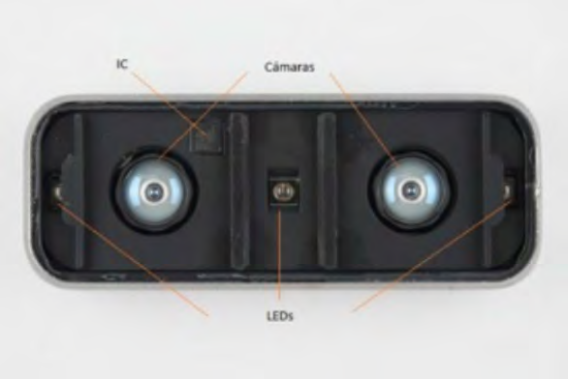


Figura 2.11: Partes de Leap Motion

Las cámaras son una de las partes fundamentales del dispositivo, porque que son las encargadas de capturar las imágenes. Cada una de estas cámaras cuenta con un sensor monocromático, sensible a la luz infrarroja, con una longitud de onda de 850 nm. Estos sensores pueden trabajar a una velocidad de hasta 200 fps, dependiendo del rendimiento del ordenador/tablet al que conectemos el dispositivo.

Los LEDs iluminan la zona de cobertura por inundación. Trabajan en el mismo espectro de luz infrarroja que las cámaras, a una longitud de onda de 850 nm. Varían su consumo eléctrico, y por tanto la iluminación, dependiendo de la luz que haya en la zona de cobertura para asegurar una misma resolución de imagen. Además, como se puede ver en la imagen 2.11, los LEDs están separados por pequeñas barreras de plástico para asegurar que la iluminación sea uniforme en toda la zona de cobertura y proteger a los sensores ópticos de una posible saturación de luz.

En cuanto al microcontrolador, éste contiene el programa que controla todo el dispositivo tanto para regular la iluminación como recoger la información de los sensores para su posterior envío al driver o controlador instalado en el ordenador. La información se envía a través de USB y soporta hasta el USB 3.0.

Por último la zona de cobertura corresponde a la mostrada en la figura 2.12 que equivale aproximadamente a una semicircunferencia de 61 cm de radio.

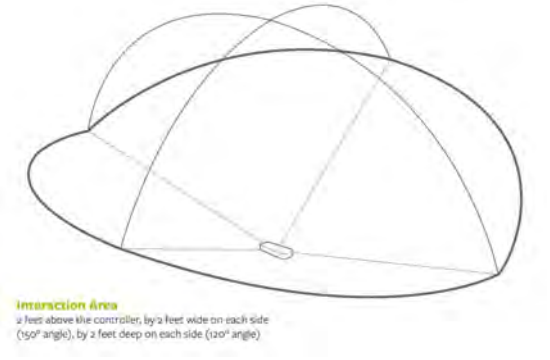


Figura 2.12: Área de interacción de Leap Motion

Sin embargo, hay que destacar que en la API del dispositivo se define una zona de trabajo llamada “Interaction Box” con un volumen de 110.55 mm de altura x 110.55 mm de anchura x 69.43 mm de profundidad, que varía sus dimensiones dependiendo de donde se encuentre el objeto a rastrear y, desde el driver del dispositivo, se puede

configurar la altura a la que se encontrará el centro que puede estar entre 7 y 25 cm desde el dispositivo, como se puede observar en la figura 2.13.

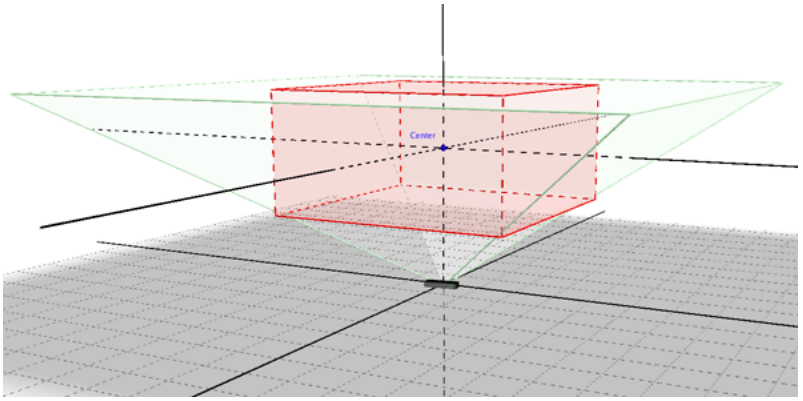


Figura 2.13: Interaction Box

2.3.2. Funcionamiento

Cuando un objeto, como una mano en este caso, es iluminado, se produce una reflexión de luz que llega al dispositivo e incide sobre las lentes de las dos cámaras. Estas lentes, de tipo biconvexas, concentran los rayos en el sensor de cada cámara; y los datos recogidos por los sensores se almacenan en una matriz (imagen digitalizada) en la memoria del controlador USB, en donde se realizan los ajustes de resolución adecuados mediante el microcontrolador del dispositivo.

Entonces una vez ajustada la resolución, ya tendriamos las imágenes recogidas por las cámaras. El proceso hasta la entrega de las variables descriptivas de una configuración de mano es algo complicado así que a continuación se expone un resumen de éste.

1. Obtiene las imágenes desde los sensores de las cámaras del dispositivo.

2. Aplica una corrección de la distorsión que producen los sensores.

3. Aplica un modelo para determinar la configuración de cada mano y ejecuta un algoritmo de visión estereoscópica entre cada par de imágenes para obtener la posición en el plano tridimensional.

2.3.3. ¿Qué información aporta?

En cuanto al tipo de datos utilizados primero hay que tener claro cuál es el sistema de coordenadas de Leap Motion así que en la figura 2.14 se puede apreciar el sistema de coordenadas cartesianas que utiliza.

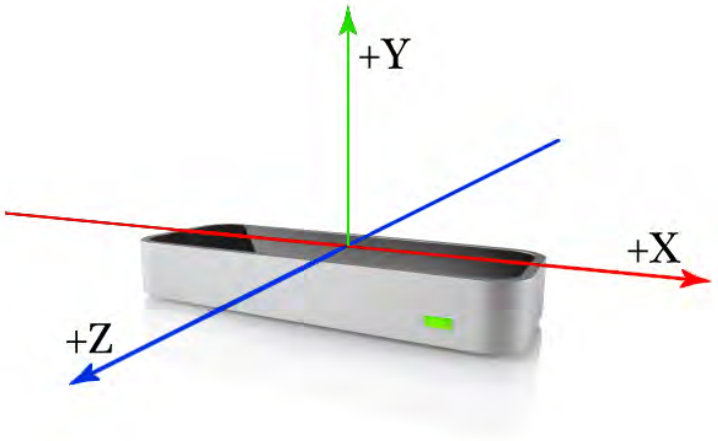


Figura 2.14: Sistema de coordenadas de Leap Motion

Además la API de Leap Motion utiliza las unidades de la tabla 2.2.

Distancia	milimetros
Time	microsegundos
Velocidad	milimetros/segundos
Ángulo	radianes

Tabla 2.2: Unidades

Respecto a la información que recoge Leap Motion, en este artículo [10], se exponen todos los datos que se extraen, tanto de velocidad como de posición o orientación, y que en la figura 2.15 se pueden apreciar todos los puntos que se monitorizan.

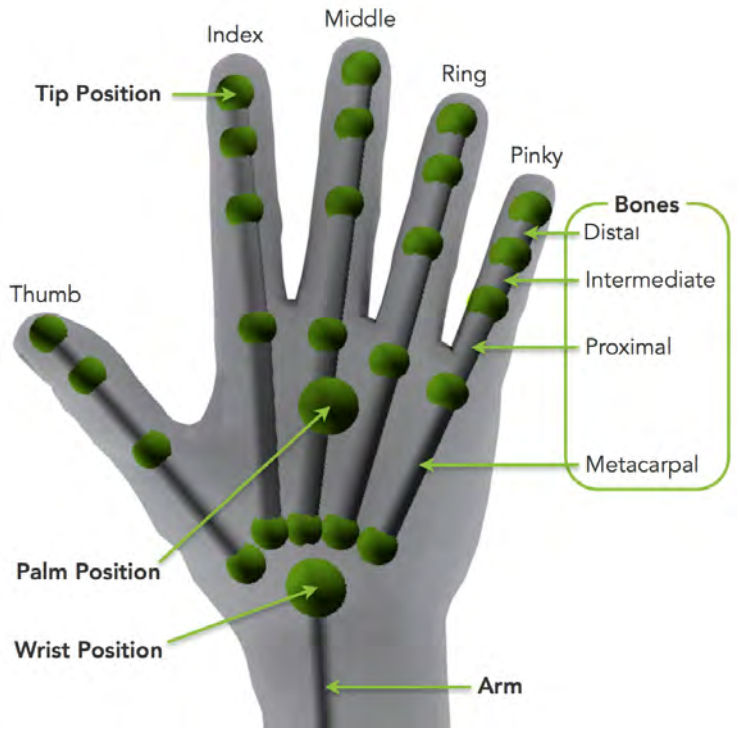


Figura 2.15: Estrucutra de la mano

Por último hay que destacar que en este proyecto la integración de Leap Motion se ha realizado a través de un package ya desarrollado que no utiliza toda la información que el dispositivo puede aportar. Este package solo transmite información, de la primera mano que detecta, de las posiciones de cada uno de los puntos de la figura 2.15, la orientación de la muñeca y la de algunos vectores importantes como el director y el normal del punto central de la mano que se encuentran definidos en un vector de tipo **leapros**, véase tabla 5.7, tal y como se explica en el package [11].

2.4. Robot AMOR

A continuación se van a abordar las características del robot AMOR que se describen en este manual [12]. Según las clasificaciones que se han expuesto anteriormente AMOR es un robot de servicio compuesto por un manipulador poliarticulado programable con un acceso de control de bajo nivel y, en este proyecto, teleoperado con 7 grados de libertad. Los diferentes eslabones de AMOR tiene nombres que coinciden con las partes del brazo como hombro (shoulder), codo (elbow) o muñeca (wrist). Además, las articulaciones también están numeradas, como se puede observar en la figura 2.16. También tiene rotación infinita en 3 de sus ejes (A1, A4 y A6), mide alrededor de 85cm y puede levantar hasta 2.5 kg de peso extra.

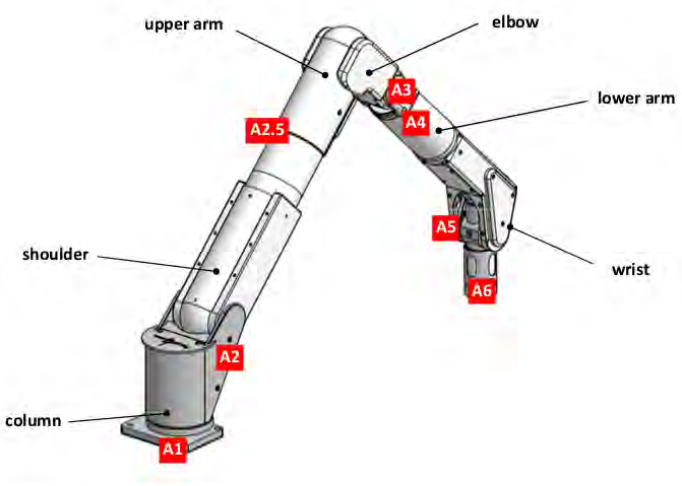


Figura 2.16: Partes del robot AMOR

Y, en la tabla 2.3, se pueden ver las características de cada articulación.

Articulación	Ratio de transmisión	Rango	Posición neutral	Límites[1]
A1	636:1	∞	0º	$-\infty/\infty$
A2	980:1	165º	-90º	-35º/130º
A2.5	594:1	300º	0º	-150º/150º
A3	396:1	154º	-90º	-88º/76º
A4	666:1	∞	0º	$-\infty/\infty$
A5	550:1	180º	180º	-180º/0º

[1] respecto a la posicón neutral de la articulación

Tabla 2.3: Características de AMOR

Por último en la figura 2.17 se pueden observar la dirección de los ángulos de cada articulación y hay que destacar que el robot está situado en la posición neutral especificada en la tabla 2.3.

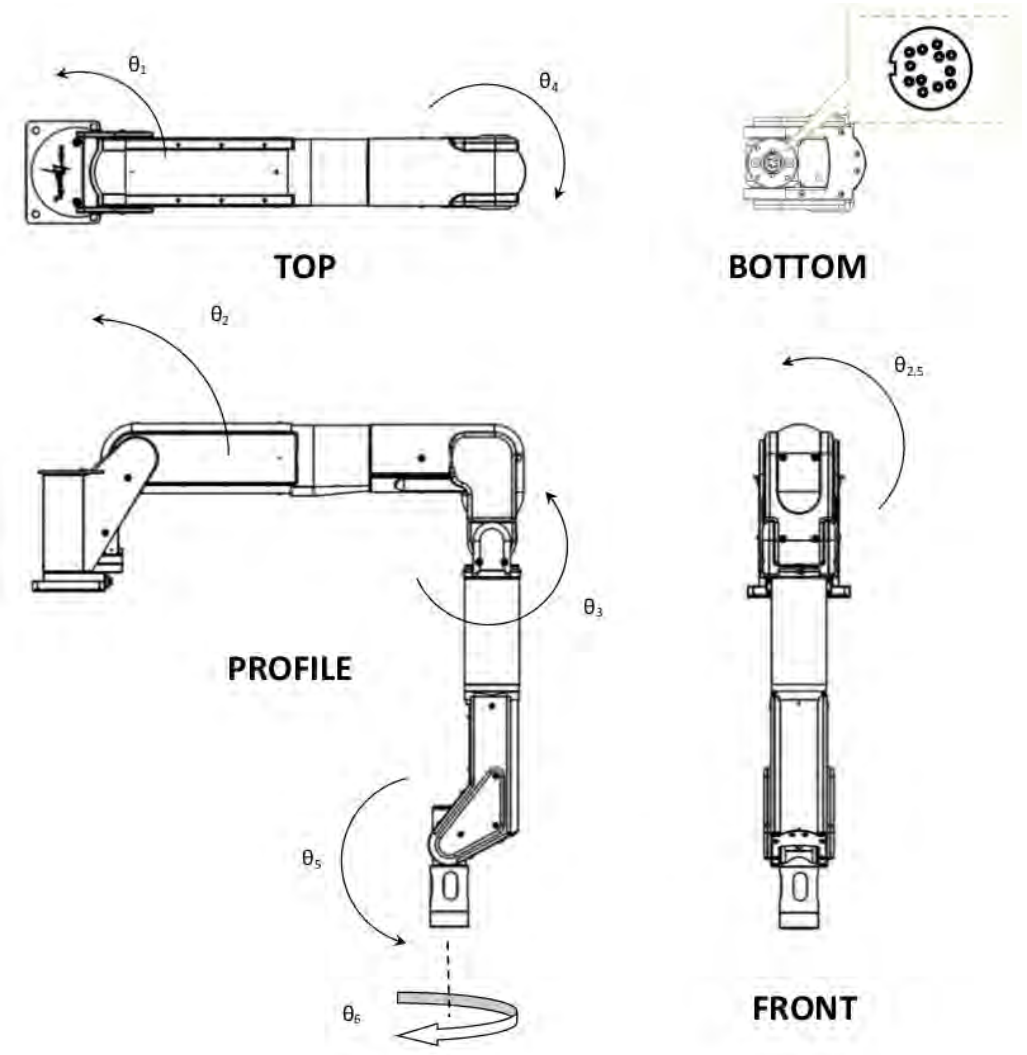


Figura 2.17: Vistas del robot AMOR

2.5. Otros trabajos relacionados

A continuación se han seleccionados varios proyectos relacionados con el tema para exponer una visión general de cada uno de ellos y conocer las ideas que han planteado así como un análisis crítico del resultado que han obtenido. Los primeros tratan básicamente sobre el reconocimiento gestual con Leap Motion mientras que los últimos se centran más en el control de algún dispositivo mediante Leap Motion.

Development of a Robotic Arm and implementation of a control strategy for gesture recognition through Leap Motion device

El objetivo de este trabajo [13] es el desarrollo de un prototipo de robot articulado es implementar un control estratégico mediante Leap Motion con el movimiento de la mano y el antebrazo. Además se propone que sea de bajo coste para que sea accesible así que la construcción del brazo articulado se realiza a partir de engranajes y piezas de impresoras o piezas imprimidas en 3D. El proyecto realiza un estudio de mercado para comparar tanto el microcontrolador a utilizar, en este caso Arduino que es la plataforma de desarrollo usada, como de los distintos sistemas de reconocimiento gestual, eligiendo Leap Motion por su gran precisión a corta distancia.

La idea de este proyecto parece muy buena porque debido a su bajo coste permitiría que, por ejemplo, instituciones educativas no universitarias poder adquirirlo para utilizarlo y enseñar tanto robótica como electrónica o mecánica desde un enfoque más práctico. Además, Arduino, que es el microcontrolador que se está usando, tiene un sistema de programación bastante sencillo lo cual facilitaría aún mas la enseñanza. Por otro lado, el brazo robótico diseñado sólo tiene 3 articulaciones lo que disminuye bastante su maniobrabilidad aunque, pero teniendo en cuenta el bajo coste del proyecto, esta deficiencia es más que asumible.

Hand Gesture Recognition with Leap Motion and Kinect devices

El objetivo de este proyecto [14] es combinar Leap Motion y Kinect para poder aumentar la precisión del reconocimiento de gestos porque exponen que Leap Motion tiene ciertos problemas con la interacción entre de dedos o cuando la mano no se encuentra en el plano paralelo al dispositivo. Asi que utilizan un algoritmo que

coteja lo obtenido por Leap y Kinect con los gestos definidos. Con Leap Motion obtiene la distancia de los dedos respecto del centro, la elevación respecto del plano de la mano y el ángulo de cada dedo respecto a la orientación de la mano. Por otro lado con Kinect recoge la curvatura de la mano y la correlación de los puntos de la mano con los gestos definidos. Así que, combinando todos estos datos, el gesto que obtenga mayor porcentaje es el se elige. Tiene una precisión cercana al 92 %.

Es bastante interesante la idea de utilizar dos dispositivos diferentes para mejorar el reconocimiento porque mientras que Leap sólo envía información de una serie de puntos, Kinect recoge la monitorización de la mano al completo. Sin embargo, como se verá a continuación, en cuanto al reconocimiento gestual se pueden obtener los mismos resultados de precisión utilizando un sólo Leap Motion.

Dynamic Hand Gesture Recognition With Leap Motion Controller

En este caso, este proyecto [15] se basa en el reconocimiento de gestos dinámicos, es decir, en movimiento. Para ello utiliza un único dispositivo Leap Motion y, a través de un algoritmo, se obtienen dos vectores. El primero es igual al que se utiliza con Leap Motion en el proyecto anterior pero el segundo informa de la interacción entre los dedos adyacentes, enviando la distancia entre ellos y el ángulo que forman. A continuación, depuran estos datos con un algoritmo llamado Hidden Conditional Neural Field (HCNF) obteniendo una precisión muy similar al del caso anterior.

El obtener una precisión similar que en el caso anterior utilizando un sólo dispositivo te da a entender que este proyecto esta mejor optimizado que el anterior. Además, en este caso son gestos en movimiento lo que, en principio, incrementa la dificultad. Por otro lado parece muy interesante el concepto del vector que maneja la interacción entre los dedos porque permite conocer fácilmente las posiciones relativas entre cada uno de ellos lo cual es vital para el reconocimiento gestual.

Hand movement and gesture recognition using Leap Motion Controller

El objetivo de este trabajo [16] es el desarrollo del reconocimiento gestual con Leap Motion tanto estáticamente como en movimiento. Para la parte estática utiliza la distancia de cada dedo con la palma y los dedos adyacentes. Por otra parte, para el reconocimiento dinámico, utiliza los vectores de velocidad de cada dedo para diferenciar entre unos movimientos y otros. Por último, se comenta que si Leap Motion monitoriza la mano correctamente, todo funciona correctamente sin embargo destaca los problemas de detección de los dedos cuando se ocultan detrás de la palma y la punta del dedo meñique y el anular que no se suelen detectar.

Aunque el diseño de este modelo es más sencillo que los anteriores parece que el planteamiento es bastante acertado. Además el uso de un elemento con la distancia entre dedos es bastante similar al utilizado en el caso anterior.

Intuitive and Adaptive Robotic Arm Manipulation using the Leap Motion Controller

Este proyecto [17] se encuentra totalmente enfocado a la robótica asistencial porque pensado para la gente de mayor edad y, en concreto a los que padecen Parkinson. Consta de un dispositivo Leap Motion, un brazo robótico JACO poliarticulado y un arduino para conectar posibles sensores externos. El control se basa en ir a al punto donde, relativamente, se encuentre la mano. Cabe destacar que se le ha añadido un mecanismo de seguridad para enfermos de Parkinson donde, cada vez que se inicia el programa, se registra un espacio de seguridad en función de lo que se mueva involuntariamente la mano del enfermo, tras una prueba de unos 15 segundos. Por último aporta un estudio de las posibles aplicaciones que pudiese tener, como por ejemplo, en la ayuda doméstica o para trabajar de forma remota.

Este es el primer proyecto de esta sección en el que se encuentra implementado el control de un brazo Robótico mediante Leap Motion. El tipo de control utilizado parece bastante intuitivo, sobre todo para la gente mayor que no suelen estar acostumbrados al uso de joysticks o teclados. Además, la idea de incluir ese mecanismo de seguridad para pacientes de Parkinson es muy interesante porque aporta cierta versatilidad al proyecto.

Home Environment Interaction via Service Robots and the Leap Motion Controller

El último proyecto estudiado [18] también se encuentra en el campo de la robótica asistencial y, al igual que en el anterior, busca mejorar las condiciones de la gente de mayor edad para incrementar su independencia. Para este proyecto se diseñó un piso experimental en donde realizar las pruebas. El proyecto está basado en ROS y tiene varios elementos conectados por wi-fi. El primero de ellos es una turtlebot con visión en 3D gracias a una Kinect y también tiene una serie de Led infrarrojos que permiten utilizar la visión nocturna para poder asistir en condiciones de baja luminosidad. Además, tiene un control de voz en base a una serie de comandos y puede controlarse de forma remota a través de Leap Motion de izquierda a derecha con el movimiento *roll* de la muñeca, y de atrás hacia delante con *pitch*. Por otro lado, también se implementa el uso de un brazo robótico JACO con el mismo método que en el trabajo anterior, el de posición. En cuanto a los resultados finales comentan que todavía están desarrollándolo y todavía no se han definido las especificaciones que va a tener ni los servicios que va a ofrecer.

Para empezar, la introducción de visión nocturna es una idea muy acertada porque permitirá al robot manejarse con seguridad cuando, por ejemplo, sea de noche y las luces estén apagadas, evitando riesgos de colisión con el usuario. El uso de comandos de voz resulta muy accesible para personas de avanzada edad mientras que el control de Leap Motion de turtlebot es bastante intuitivo y no requiere grandes desplazamientos de la mano. Respecto al brazo robótico, al igual que en el caso anterior, el método de control por posición parece muy adecuado para el perfil de usuario al que está dirigido este proyecto. Por último, el empleo de ROS es muy interesante porque, al estar basado en un diseño modular, permite añadir o eliminar elementos al sistema de una forma bastante sencilla.

Capítulo 3

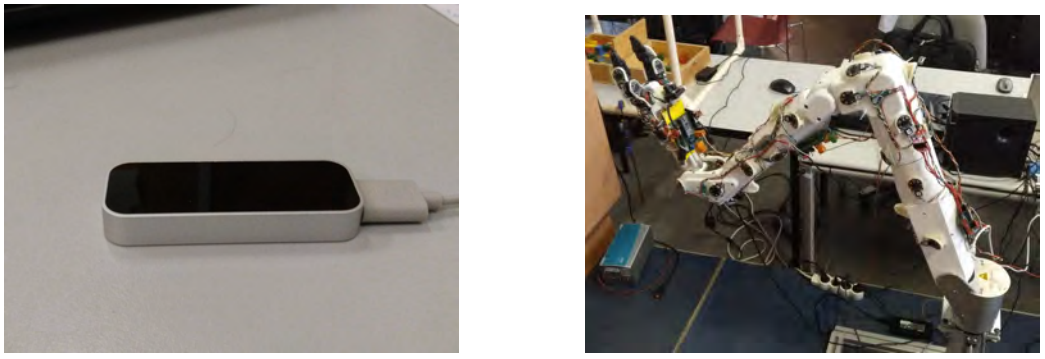
Desarrollo del TFG

Índice

3.1. Visión general

En este capítulo abordaremos todo lo relacionado con el proceso de creación del proyecto. El trabajo se ha desarrollado en el sistema operativo **Linux**, en la version de **Ubuntu 16.04 LTS**, al no estar disponible ROS para Windows, y en concreto vamos a utilizar la versión de ROS llamada **kinetic**.

El proyecto consiste en la creación de un **sistema de control sin contacto** basado en **ROS** para el **brazo robótico AMOR** mediante del uso de un dispositivo **Leap Motion**. Primero se obtendrá la información que extrae Leap Motion de la mano y el antebrazo en un módulo de ROS para que luego poder publicarla a través de los topic de ROS, que serán la base de la comunicación de nuestro sistema de control.



(a) Leap Motion (b) robot AMOR

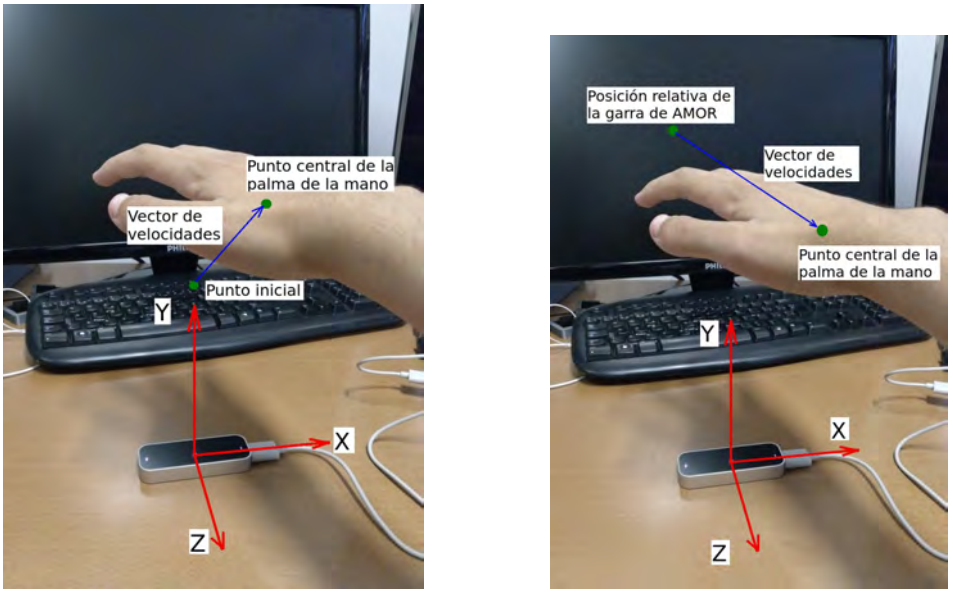
Figura 3.1: Dispositivos utilizados

A continuación se diseñarán el programa o los programas que se vayan a utilizar y, para terminar, se establecerá una vía de comunicación con AMOR. El programa de control de AMOR permite mover el robot tanto en el plano cartesiano como en el articular e incluye el control de la apertura y cierre de la garra. Además, está basado en YARP, una plataforma similar a ROS, por lo que se deberá encontrar algún método de comunicación entre ambas plataformas.

3.1.1. Hipótesis planteadas

Al comienzo de este proyecto, se trazaron una serie de ideas como estrategia inicial y, a partir de las cuales, comenzar a trabajar. No eran definitivas ni mucho menos, pero son la base del resultado final.

La primera idea a tener en cuenta era el **tipo de control**, el cómo se iba a diseñar. Hay que destacar que el programa de AMOR se mueve a través de vectores de velocidad así que, sin olvidar esta condición, se plantearon dos formas distintas de control.



(a) Control de joystick (b) Control de posición

Figura 3.2: Tipos de control

El primero era un tipo de **control de joystick**, figura 3.2a, donde se escogería un punto en el espacio de Leap Motion como inicial y dependiendo de donde esté situada la mano, el vector que formarían el punto central de la mano y ese punto inicial correspondería proporcionalmente con el vector de velocidad que se enviaría a AMOR. Este ha sido el que se planteó en los trabajos [17] y [18] y es similar al sistema utilizado en la parte de Turtlebot de [18].

El segundo era un tipo de **control de posición** y consiste en que el robot vaya a al posición relativa donde se encuentre la mano, como se puede ver en la figura

3.2b. Por tanto, hay que diseñar un espacio común de coordenadas entre AMOR y Leap Motion. Además, como sólo se pueden enviar velocidades y no puntos, el método necesita recibir desde AMOR la posición relativa de la garra, y, junto con el punto central de la mano, se trazaría el vector de velocidades que hay que enviar. Este método es similar al que se usa en [13] y en la parte del control del robot JACO en [18].

También se planteó la idea de utilizar dos Leap Motion a la vez de manera que uno manejase las velocidades lineales de AMOR mientras que el otro se encargase de controlar la orientación de la garra del robot y de los gestos encargados de controlar la apertura y cierre de la ésta u otras acciones concretas. Por otro lado, estaba la opción de utilizar un único Leap Motion donde se realizarían tanto las tareas de reconocimiento gestual como el control de velocidades.

En cuanto a la comunicación entre ROS y Leap Motion se decidió utilizar el package llamado rosleapmotion [11] así que sólo hay que crear un espacio de trabajo en ROS donde instalar ese package siguiendo las instrucciones que vienen incorporadas.

Por último, en la página web *http://www.yarp.it/yarp_with_ros.html* aparecen varios métodos de comunicación entre YARP y ROS así que se probarán y se escogerá el más conveniente.

3.2. Fases del proyecto

A continuación, se van a tratar las distintas fases de este proyecto. Al explicar detalladamente el desarrollo del mismo, se otorgará una idea aproximada de los razonamientos utilizados durante su realización y será bastante esclarecedor a la hora de discernir el por qué se ha utilizado un planteamiento u otro.

Además, esta repartición permitirá facilitará la explicación de cada uno de los elementos de este proyecto. El proyecto se ha compuesto por 5 fases y la solución final y la figura 3.3, se puede observar un esquema de las distintas fases donde, se indican los requisitos previos de cada una de ellas para comenzar con su desarrollo.

Figura 3.3: Fases del proyecto

3.2.1. Integración del Leap Motion en ROS

El objetivo de esta fase es la implementación de Leap Motion en ROS. Para ello se ha utilizado un package, ya desarrollado, que se encuentra en la wiki de ROS en el este enlace [11]. Este package sólo permite extraer información de la mano que detecta primero, a pesar de que Leap Motion es capaz de reconocer ambas manos. A la hora de la instalación hay que seguir los pasos que se encuentran en el archivo Readme.txt que viene incluido dentro.

Este package fue realizado por Florian Lier y Mirza Shah e incluye la adaptación de Leap Motion a un módulo de ROS y un programa ejecutable, llamado sender.py, que publica directamente toda la información que recibe Leap Motion en el topic de ROS llamado */leapmotion/data*. Además han creado un tipo de dato, llamado **leapros** (tabla 5.7), que engloba todos los datos obtenidos de la mano.

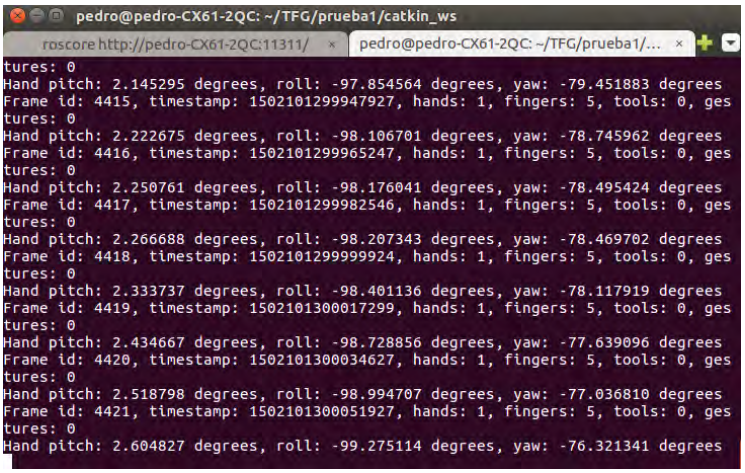


Figura 3.4: Sender.py

Además de publicar información, también muestra cuantas manos y cuantos dedos detecta, a pesar de solo publicar en ROS una sola mano, y el valor de la orientación de la mano que se esté detectando, es decir, roll, pitch y yaw. En la figura 3.4 se puede observar el output que muestra el programa sender.py al ejecutarse.

Por otro lado, a la hora de ejecutar el programa, el dispositivo Leap Motion debe estar conectado al ordenador y en Ubuntu hay que inicializarlo en un terminal a través del comando *sudo leapd*. Y si quieres modificar los ajustes de Leap Motion (figura 3.5), en Ubuntu sólo hay que ejecutar el comando *sudo LeapControlPanel*,

que mostrará un icono de Leap Motion en la parte superior derecha de la pantalla donde se podrá acceder a estas funciones.

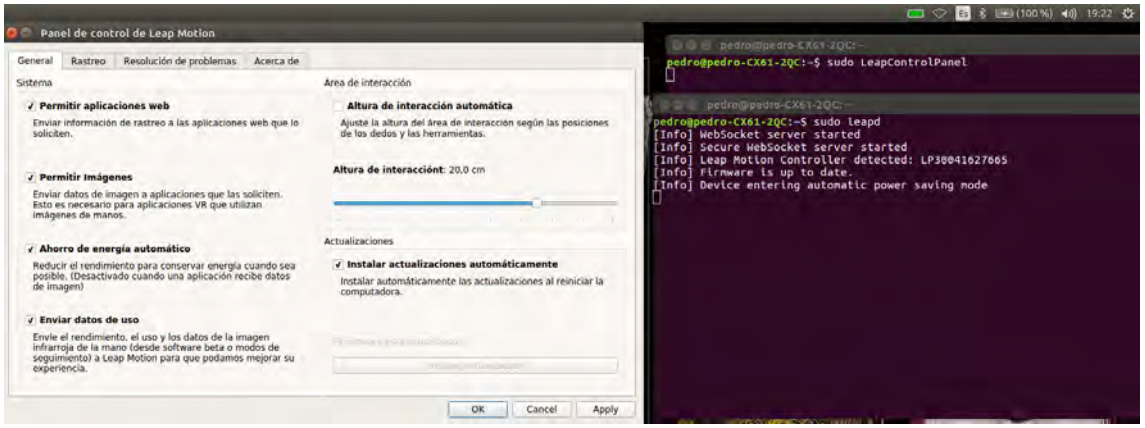


Figura 3.5: Panel de control de Leap Motion

Además la interfaz de los ajustes lleva incluido un visualizador (figura 3.6) que muestra las manos detectadas por Leap Motion,lo cual es bastante útil a la hora de comprobar si se está detectando correctamente la mano.

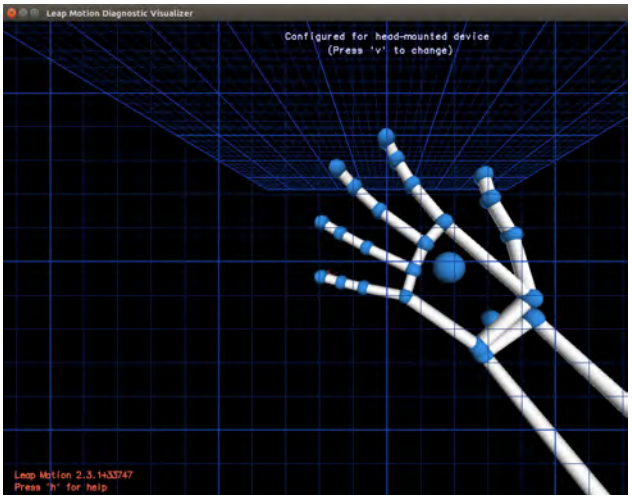


Figura 3.6: visualizador

Entonces, una vez instalado el package en un workspace de ROS, se diseñó un programa, cuyo nombre es *subscription.cpp*, que se subscribiese al topic anteriormente citado. En la figura 3.7 se puede ver un esquema de las conexiones entre los distintos programas.

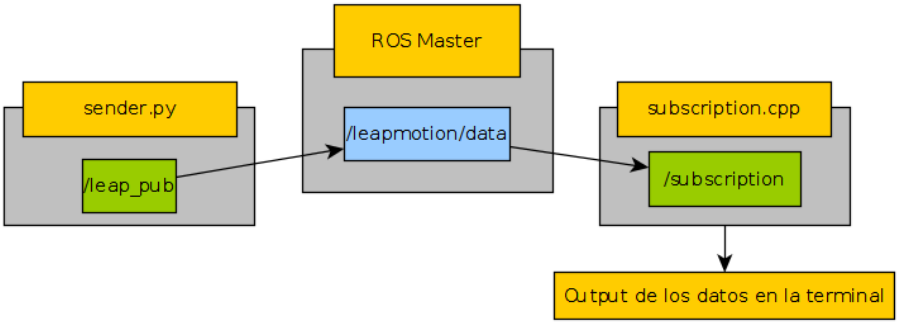


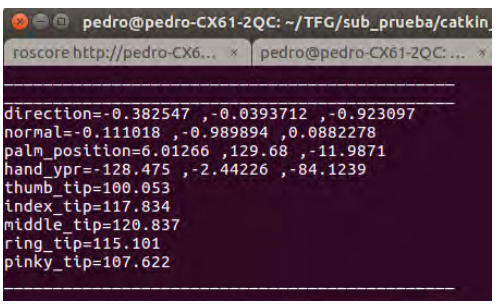
Figura 3.7: Esquema de las conexiones (I)

Por lo tanto, este programa recibe la información del topic */leapmotion/data* y la guarda en una varible de tipo **leapros** (ver tabla 5.7) llamada data. Además, muestra en la terminal los datos de la mano sobre el punto central, el vector director y normal, los ángulos roll, pitch y yaw y un vector, dedos[5], con cinco componentes indicando la distancia desde la punta de cada dedo al centro de la mano.

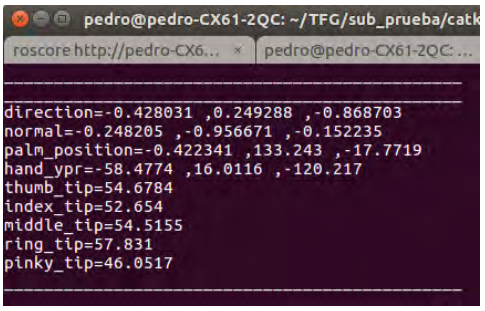
Este vector, dedos[5], se obtiene al utilizar la ecuación (3.1) donde el punto (x1,y1,z1) corresponde al punto central de palma de la mano y el punto (x2,y2,z2) a la punta del dedo correspondiente.

$$dedos(i) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{3.1}$$

Así que, una vez ejecutado el programa, muestra en el terminal lo que se puede ver en la figura 3.8, donde se puede apreciar la información anteriormente descrita. Se han recogido datos de la mano en dos posiciones para poder observar la diferencia de valores de los dedos obtenidos en la ecuación (3.1) en función de si está abierta la mano, figura 3.8a, o si está cerrada, figura 3.8b.



(a) Mano abierta



(b) Mano cerrada

Figura 3.8: subscription.cpp

3.2.2. Primeras pruebas de control con Leap Motion

Respecto a las hipótesis propuestas en la sección 3.1.1, se eligió probar primero el control de posición porque parece mas intuitivo y cómodo que el uso de un joystick para las personas que no hayan tenido demasiado contacto con este tipo de sistemas ya que, en teoría, mueves la mano al punto a donde quieres que vaya el robot.

Así que en esta fase toca el diseño de un primer programa de control de prueba. Para esta prueba se eligió el simulador turtlesim que viene incluido en ROS. Este programa simula en 2 dimensiones, como se puede observar en la figura 3.9, una tortuga dentro del recuadro azul cuyo centro de coordenadas reside en la esquina inferior izquierda.

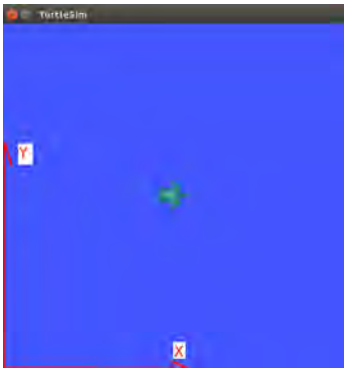


Figura 3.9: Turtlesim

En cuanto a su movimiento, turtlesim está suscrito al topic */turtle1/cmd_vel* y para mover la tortuga hay que enviar el módulo del vector de la velocidad y un ángulo de disparo en z como en la figura 3.12. A su vez, turtlesim publica en el topic */turtle1/pose* la posición de la tortuga y su orientación en cada momento usando un tipo de dato de turtlesim llamado **Pose** con tres componentes: Pose.x, Pose.y y Pose.theta. En la figura 3.10 hay un esquema de las comunicaciones de esta fase.

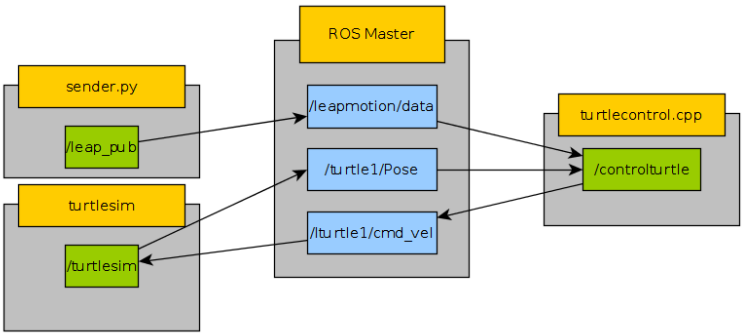


Figura 3.10: Esquema de las conexiones (II)

El programa de control, *turtlecontrol.cpp*, está subscrito a la información de Leap Motion y de la posición de la tortuga. Mientras tanto publica las órdenes de movimiento para el control de la tortuga.

Al haber sólo dos dimesiones sobraba un eje en los datos que se obtienen de Leap Motion, figura 2.14, por lo que se decidió despreciar el eje vertical, que corresponde con el eje **y**, para manejar la tortuga en un plano horizontal al suelo,es decir, los ejes **x** y **z**. Por lo tanto los ejes quedarían los de la figura 3.11.

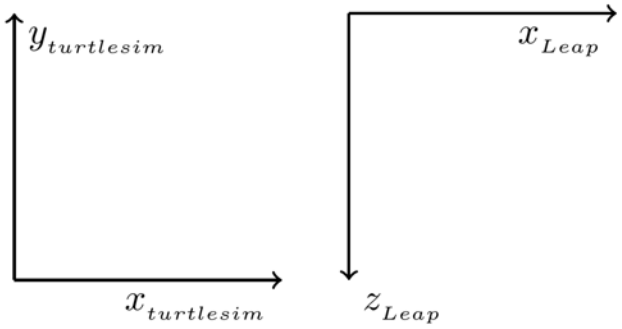


Figura 3.11: **Izq.** Turtlesim **Dcha.** Leap Motion

Primero hay que estudiar las dimensiones de cada uno para poder utilizar un espacio común de coordenadas:

- **Turtlesim**. Como se ha mencionado antes, el origen está en la esquina inferior izquierda así que la longitud del lado se corresponderá con las dimensiones de ambos ejes y el rango es de 0 a 11.1 ud.
- **Leap Motion**. En este caso el centro de coordenadas se encuentra en el centro del Leap motion así que se decidió que tanto en **x** como en **z**, despreciando **y** que es el eje vertical, se utilizara un rango de -150 a 150 unidades.

Entonces, como el espacio de Turtlesim comienza en 0, hay que añadir un offset de 150 unidades positivas en el espacio de Leap Motion para utilizar un rango únicamente compuesto por números positivos, que en este caso será de 0 a 300. Después se multiplicará el espacio de Leap Motion por 11.1 (rango máximo de Turtlesim) al mismo tiempo que se dividirá entre 300 (rango máximo de Leap Motion) para adaptarlo completamente al espacio de coordenadas de Turtlesim.

$$X_{turtlesim} = \frac{11{,}1 \times (150 + X_{Leap})}{300} \quad Y_{turtlesim} = \frac{11{,}1 \times (150 - Z_{Leap})}{300} \tag{3.2}$$

Por lo tanto nos quedará la ecuación (3.2) donde hay resaltar el signo negativo en el eje Z de Leap debido a que estaba orientado hacia la dirección opuesta (figura 3.11).

Así que, una vez diseñado un espacio común de coordenadas, se procedió al diseño del control de la tortuga. La estrategia a seguir fue crear un vector a partir de las coordenadas de la tortuga y las coordenadas del centro de la mano, que es a donde queríamos que fuese la tortuga. A continuación se utilizará el módulo este vector para la velocidad y un ángulo de disparo respecto al ángulo de la orientación actual de la tortuga que también se recibe a través del topic*/turtle1/pose*, figura 3.12.

Figura 3.12: Obtención de V_cmd y theta

Entonces las ecuaciones resultantes serían (3.3) y (3.4) donde K es una constante añadida para adecuar el comportamiento del programa . Además los datos de la mano ya han sido transformados y adaptados al nuevo sistema de coordenadas.

$$v_{cmd} = K_v \times \sqrt{(x_{turtle} - x_{mano})^2 + (y_{turtle} - y_{mano})^2} \tag{3.3}$$

$$\theta_{cmd} = K_\theta \times (\arctan(\frac{y_{turtle} - y_{mano}}{x_{turtle} - x_{mano}}) - \theta_{turtle}) \tag{3.4}$$

Así que, esta información se publicaría en el topic */turtle1/cmd_vel* cada 0.1 segundos y hay que destacar que si la tortuga está a menos de 1 unidad de distancia del punto de la mano se pone a 0 la velocidad y el ángulo de disparo porque, cuando estaban demasiado cerca, el movimiento era muy inestable y no conseguía alcanzar el punto.

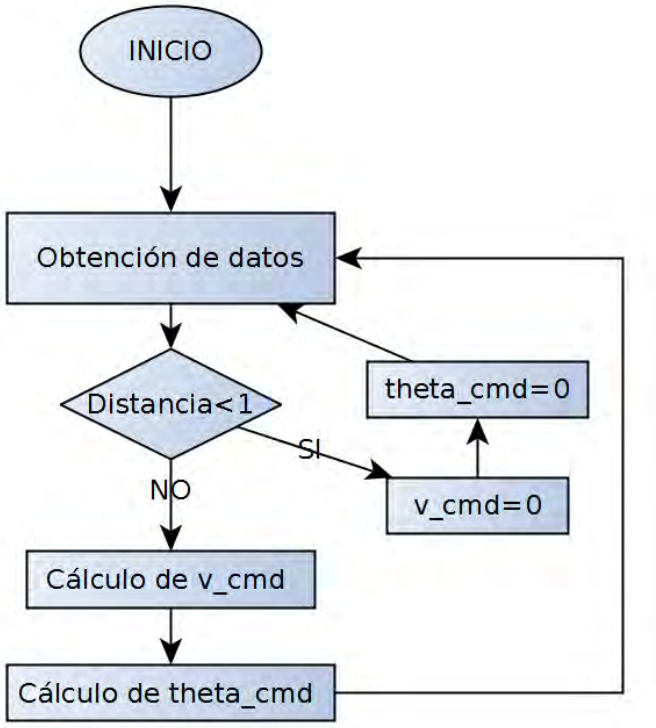


Figura 3.13: Diagrama de turtlecontrol.cpp

Además habría que incluir tanto en el programa, llamado turtlecontrol, como en package.xml del workspace las librerías que aparece en la figura 3.14.

```
#include <ros/ros.h>
#include <leap_motion/leapros.h>
#include <turtlesim/Pose.h>
#include <geometry_msgs/Twist.h>
#include <math.h>
```

Figura 3.14: Librerías de turtlecontrol.cpp

Y el código de turtlecontrol.cpp quedaría tal y como se puede observa en la figura 3.15. Hay que destacar 2 comandos que se encuentran justo debajo del inicio del bucle principal,*while(ros::ok())*, que son *ros::Rate(10).sleep* y *ros::spinOnce()*.

El primero sirve para ajustar la frecuencia (Hz) ejecución del bucle y el segundo habilita una única ejecución de las funciones enlazadas con las subscripciones así que cada vez que se de una vuelta dentro del bucle se ejecutara ese comando que permitirá la adquisición de datos, como viene en la figura 3.13.

```
geometry_msgs::Twist palm,msg;
turtlesim::Pose turtle;

void palmposMR(const leap_motion::leapros& msg) ;
void turtleposeMR(const turtlesim::Pose& msg);
inline double getdistance(double x1, double y1, double x2, double y2);

int main(int argc, char **argv){
// inicializacion de los nodos
        ros::init(argc, argv, "controlturtle");
        ros::NodeHandle np;
        ros::NodeHandle ns1;
        ros::NodeHandle ns2;
//publicacion en cmd_vel
        ros::Publisher pub=np.advertise<geometry_msgs::Twist>("turtle1/cmd_vel",100);
//subscripcion a leap motion
        ros::Subscriber sub1= ns1.subscribe("leapmotion/data", 100, &palmposMR);
//subscripcion a turtle pose
        ros::Subscriber sub2= ns2.subscribe("turtle1/pose",100,&turtleposeMR);

        while (ros::ok()){
        ros::Rate(10).sleep();
        ros::spinOnce();

        if( getdistance(palm.linear.x, palm.linear.y, turtle.x, turtle.y) > 1 ){
        msg.linear.x=1.5*getdistance(palm.linear.x, palm.linear.y, turtle.x, turtle.y);
        msg.linear.y=0;
        msg.linear.z=0;
        msg.angular.x=0;
        msg.angular.y=0;
        msg.angular.z=4*(atan2(palm.linear.y - turtle.y, palm.linear.x - turtle.x) - turtle.theta);
        }else{
        msg.linear.x=0;
        msg.linear.y=0;
        msg.linear.z=0;
        msg.angular.x=0;
        msg.angular.y=0;
        msg.angular.z=0;
        }
        //ROS_INFO_STREAM("mod_v="<<msg.linear.x<<" ,w="<<msg.angular.z<<", theta_turtle="<<turtle.
        pub.publish(msg);
}
}
void palmposMR(const leap_motion::leapros& msg) {
        palm.linear.x=11.1*(150+msg.palmpos.x)/300;
        palm.linear.y=11.1*(150-msg.palmpos.z)/300;
}
void turtleposeMR(const turtlesim::Pose& msg){
        turtle.x=msg.x;
        turtle.y=msg.y;
        turtle.theta=msg.theta;
}
inline double getdistance(double x1, double y1, double x2, double y2){
        return sqrt(pow(x1-x2,2)+pow(y1-y2,2));
}
```

Figura 3.15: turtlecontrol.cpp

3.2.3. Comunicación entre ROS y AMOR

Para poder controlar la comunicación con AMOR primero había que encontrar la forma de comunicar ROS con YARP, que es la plataforma en la que utiliza así que esta fase se centra en realizar la conexión entre ambas plataforma. Como se comentó al principio en la página web *http://www.yarp.it/yarp_with_ros.html* exponen varios métodos.

Según esa web, hay una serie de comandos en YARP que son capaces de publicar y suscribirse en ROS pero primero hay que enlazar ambas plataformas. El funcionamiento de ambas es similar dado que YARP necesita crear primero un servidor para luego poder ejecutar sus programas, como ROS con ROS master. Este servidor se crea al escribir en el terminal el comando *yarpserver* y la clave para enlazar ambos servidores se basa en ejecutar primero roscore en un terminal y luego en otro ejecutar el comando *yarpserver --ros* que crea el servidor de YARP y lo enlaza con roscore. En la figura 3.16 se ha señalizado el fragmento donde muestra la conexión.

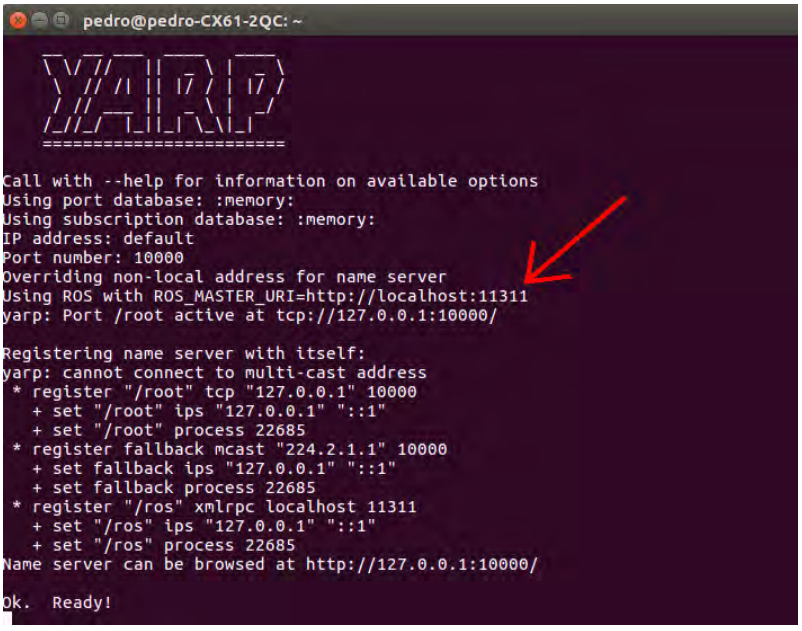


Figura 3.16: yarpserver

Por lo tanto, tras enlazar los servidores, comenzaron las pruebas para lograr la comunicación entre plataformas. En primer lugar vamos a explicar cómo suscribirse a topics de ROS desde YARP que es la base de nuestra comunicación.

Como sucede en ROS, YARP necesitaría declarar un nodo para poder utilizar los topics de ROS y enlazarlo con los topics en los que quieran publicar o suscribirse utilizando el código en la figura 3.17. Además, al igual que en ROS, se debe que declarar el tipo de dato que se va a usar.

```
yarp::os::Node node1("/yarp/listener");
yarp::os::Subscriber<Twist> subscriber;

 if (!subscriber.topic("/chatter")) {
    cerr<< "Failed to subscriber to /chatter\n";
    return -1;
}
yarp::os::Node node2("/yarp/talker");

yarp::os::Publisher<Twist> publisher;

 if (!publisher.topic("/posicion2")) {
    cerr<< "Failed to create publisher to /posicion2\n";
    return -1;
}
```

Figura 3.17: Declaración de nodos en YARP y conexión con los topics de ROS

Tras haber hecho varias pruebas, se ha llegado a la conclusión de que en YARP es recomendable utilizar un solo nodo por conexión porque si no surgen algunos errores de comunicación, a pesar de que en ROS no es necesario.

Entonces sólo queda la extracción o el envío de los datos correspondientes mediante el uso de una variable, del mismo tipo de dato que se ha declarado en el fragmento de la figura 3.17, para guardar estos datos. Siguiendo con el ejemplo anterior, en la figura 3.18 se ejecuta la lectura de un subscripción que guarda los datos en la variable *rdata* mientras que se publican los datos que se encuentran en la variable *sdata*.

```
while (true) {
    subscriber.read(rdata);
    cout << "Received:" <<
    publisher.write(sdata);
}
```

Figura 3.18: Extracción y envío de datos

A continuación se puede ver en la figura 3.19 el esquema de la comunicación de los programas de esta fase. No se va a añadir un diagrama del funcionamiento del programa utilizado porque es demasiado simple, se basa en un bucle que recibe un dato y lo muestra en la terminal mientras que publica otro dato.

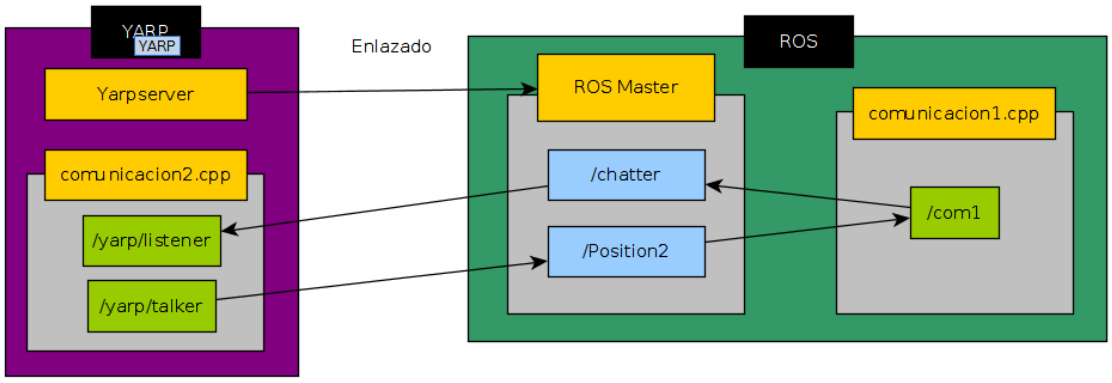


Figura 3.19: Esquema de las conexiones (III)

Entonces, una vez diseñado el programa,se ejecutó el programa y el resultado está en la figura 3.20 donde se está enviando un vector con tres ceros desde ROS a YARP, que se muestra por pantalla en la parte izquierda, mientras se envía desde YARP a ROS otro vector aunque en este caso se compone por tres dieces, tal y como aparece en el terminal situado más a la derecha.

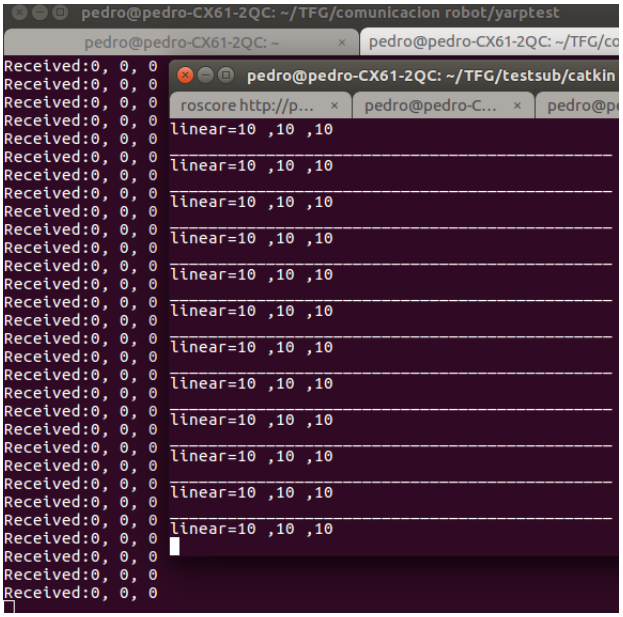


Figura 3.20: Prueba de comunicación entre ROS y YARP

3.2.4. Diseño del sistema de control de AMOR

A continuación, después de haber logrado la comunicación entre ambas plataformas y la conexión con Leap Motion, se inició el diseño de un primer programa para el control del robot AMOR a partir del programa utilizado en el apartado 3.2.2 aunque, esta vez, el sistema necesitará los tres ejes cartesianos. En cuanto a las hipótesis planteadas en la sección 3.1.1 se utilizó control de posición y un solo Leap Motion pero, en el caso de que hubiese problemas, se valoraría la introducción de otro.

Al igual que con Turtlesim, se empleó el método de control de posición. Primero se implementó el control de las velocidades lineales y, más adelante, se estudiaría la introducción de algun tipo de control para las velocidades angulares. Por otro lado, hay que destacar que en esta fase hay dos partes relativamente distintas: la primera concierne al programa basado en ROS donde se reciben y transforman los datos tanto de Leap Motion como los de AMOR para calcular las velocidades lineales correspondientes, mientras que la segunda parte tiene que ver con la adaptación del programa de AMOR para que pueda publicar y suscribirse a ROS.

El esquema de las conexiones de este sistema aparece en la figura 3.21 donde se pueden ver 3 topics distintos: */leapmotion/data*, */posicion2* y */chatter*.

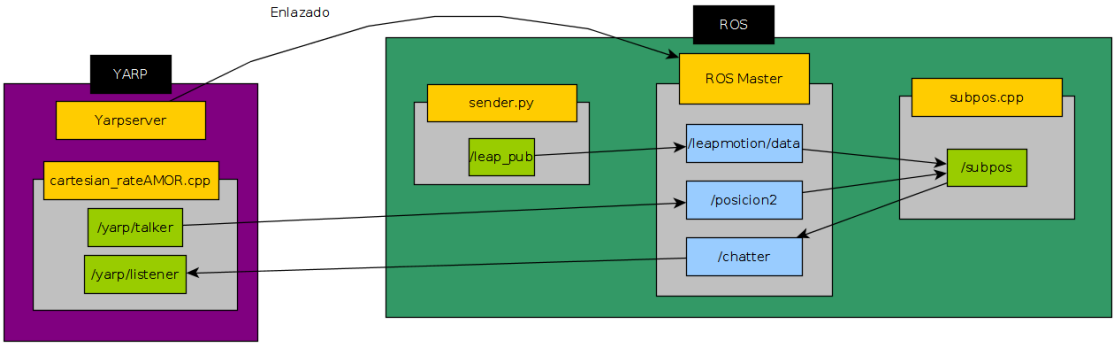


Figura 3.21: Esquema de las conexiones (IV)

El primero, */leapmotion/data*, funciona igual que en las fases anteriores, es decir, guarda los datos procedentes de Leap Motion. Los dos siguientes se utilizan para la comunicación entre el programa de control de ROS y el programa de YARP de AMOR. En */posicion2* está la posición actual de la garra en cartesianas y en */chat-*

ter se encuentran los valores de velocidad con los que se mueve el robot.

Para empezar, es importante tener claro la orientación tanto de AMOR como la de Leap Motion para así poder crear un espacio común en el que poder trabajar. En la figura 3.22 se encuentran los ejes utilizados por AMOR mientras que los de Leap Motion se corresponden con los de la figura 2.14.

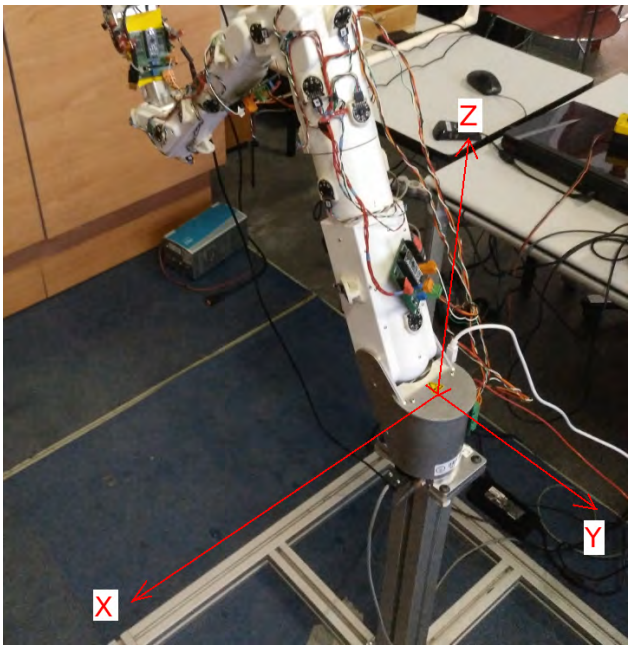


Figura 3.22: Robot AMOR

Y, a continuación en la figura 3.23 se compararán los ejes de AMOR y Leap Motion y los que se van a usar en el programa de control para las velocidades.

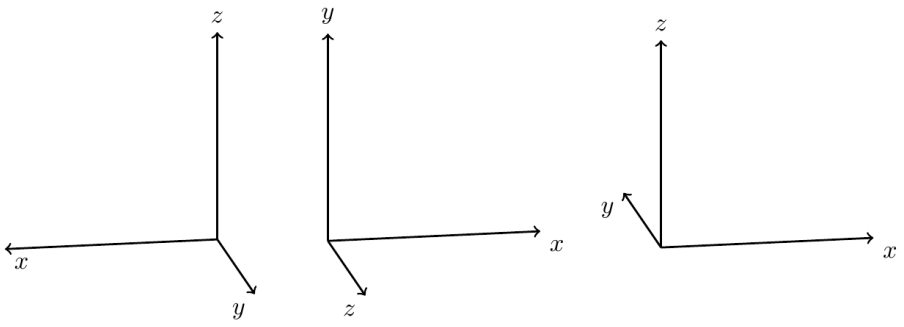


Figura 3.23: **Izq** Robot AMOR, **Centro** Leap Motion y **Dcha** Ejes de las velocidades

Diseño del programa de ROS

En cuanto al programa de control, una vez acordado el espacio de coordenadas, tiene que obtener los datos de AMOR y de Leap Motion, es decir, suscribirse a los topics correspondientes, calcular las velocidades lineales correspondientes y publicarlas en un topic para enviarlas a AMOR así hay que crear un nodo, llamado en este caso **subpos**, para poder realizar esas conexiones, como se puede observar en la figura 3.24. Al principio se inicializa el nodo y luego el comando *ros::NodeHandle* nos permite crear un elemento que enlaza este nodo con el topic correspondiente del esquema de la figura 3.21, ya sea como publicista o subscriptor.

```
ros::init(argc, argv, "subpos");
ros::NodeHandle nh;
ros::Subscriber sub = nh.subscribe("posicion2", 1000, &AMORpos);
ros::NodeHandle nl;
ros::Subscriber sub2 = nl.subscribe("leapmotion/data", 1000, &palmposMR);
ros::NodeHandle ns;
ros::Publisher pub=ns.advertise<geometry_msgs::Twist>("chatter",1);
```

Figura 3.24: Nodos utilizados para la primera prueba

El diagrama de la figura 3.25 corresponde con el funcionamiento de este programa. Básicamente adquiere los datos mediante la ejecución de las subscripciones que están anotadas en el cuadro verde y la parte del cálculo de la velocidad y su publicación que son los bloques que están rodeados con un cuadro amarillo claro.

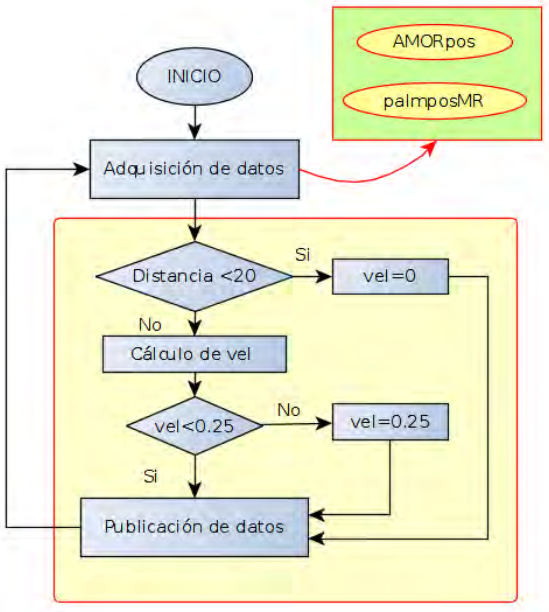


Figura 3.25: Diagrama de subpos.cpp

En la figura 3.26 se encuentra el diagrama de la ejecución de las susbcripciones para la adquisición de los datos.

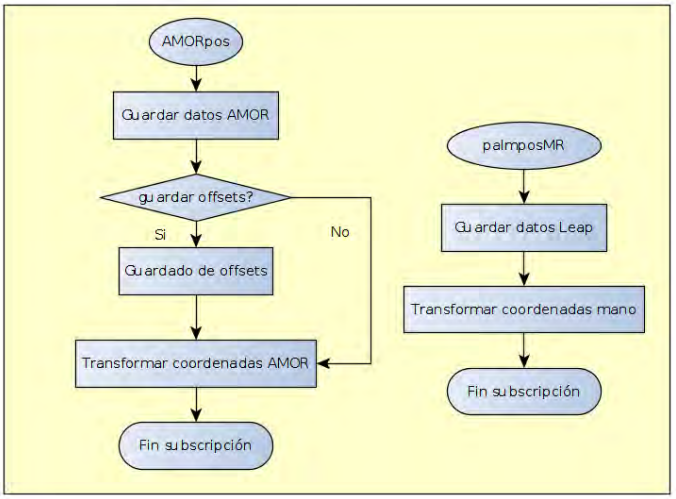


Figura 3.26: Ejecución de las subcripciones de la primera prueba con AMOR

La función **palmposMR** tiene que ver con la obtención de los datos de Leap Motion donde simplemente se transforman al espacio de coordenadas utilizado y se guardan, sin embargo, **AMORpos** que se corresponde con la adquisición de la posición de AMOR es algo mas complicada. La razón es que la posición de la garra de AMOR no se encuentra en en el punto (0,0) del plano horizontal, como ocurre con Leap Motion, así que primero hay que guardar un offset para forzar que su posición inicial en ese plano esté en el punto (0,0) y esto sólo puede suceder la primera vez que se pase por esta subscripción para luego tener esos offset en cuenta a la hora de transformar los datos al nuevo espacio de coordenadas.

En cuanto al cálculo de la velocidad se usó el método de control de posición, ver la figura 3.2b, del que se ha hablado anteriormente. Primero se estudia si la distancia entre la mano y AMOR supera la mínima, para evitar los problemas que se detectaron con turtlesim en la sección 3.2.2, y si es así se guarda el vector formado. Se le ha añadido un tope de velocidad para evitar valores demasiado elevados

Adaptación del programa de AMOR

En cuanto al programa de AMOR se tuvo en cuenta lo que se investigó en la fase anterior. Por lo tanto, primero hay que descubrir la parte del programa donde se encuentra el control de velocidad o la posición del robot para luego poder enviarlo o modificarlo.

Una vez revisado hay dos fragmentos, figura 3.27, donde aparecen el control de velocidad de AMOR (figura 3.27b) y la publicación de la posición de la garra del robot (3.27a).

```
// get cartesian positions
AMOR_VECTOR7 cartesianPositions;
if(amor_get_cartesian_position(g_hRobot, cartesianPositions) != AMOR_SUCCESS) {
        HandleError();
}
yarp::os::Bottle& bottle_x_o = port_x_o.prepare();
bottle_x_o.clear();
bottle_x_o.addString("x");
bottle_x_o.addDouble(MM2M(cartesianPositions[0]));
bottle_x_o.addString("y");
bottle_x_o.addDouble(MM2M(cartesianPositions[1]));
bottle_x_o.addString("z");
bottle_x_o.addDouble(MM2M(cartesianPositions[2]));
bottle_x_o.addString("roll");
bottle_x_o.addDouble(RAD2DEG(cartesianPositions[5]));
bottle_x_o.addString("pitch");
bottle_x_o.addDouble(RAD2DEG(cartesianPositions[4]));
bottle_x_o.addString("yaw");
bottle_x_o.addDouble(RAD2DEG(cartesianPositions[3]));
port_x_o.write();
```

(a) Publicación de la posición

```
// receive Cartesian rate
yarp::os::Bottle *_bottleReceived = port_dx_i.read(false);
if (_bottleReceived!=NULL) {

        // not throwing error if tag does not exist
        checkBottleForDouble(_bottleReceived,"dx",cartesianRateReceived.at(0));
        checkBottleForDouble(_bottleReceived,"dy",cartesianRateReceived.at(1));
        checkBottleForDouble(_bottleReceived,"dz",cartesianRateReceived.at(2));
        checkBottleForDouble(_bottleReceived,"droll",cartesianRateReceived.at(3));
        checkBottleForDouble(_bottleReceived,"dpitch",cartesianRateReceived.at(4));
        checkBottleForDouble(_bottleReceived,"dyaw",cartesianRateReceived.at(5));

        //printf("_dx, _dy, _dz: %f, %f, %f.\n", cartesianRateReceived.at(0), cartesia
}
```

(b) Control de velocidades cartesianas

Figura 3.27: cartesian_rate_ AMOR.cpp

El fragmento 3.27a corresponde con la publicación de la posición de la garra en un *bottle* de YARP, que es un elemento de YARP que guarda información y es la clave de la comunicación de esta plataforma. Entonces lo que hay que hacer es guardar esa información en una variable del tipo correspondiente y enviarla a través

de ROS, tal y como hemos hecho en el apartado anterior. En el fragmento 3.27b se encuentra el control de la velocidad cartesiana por lo tanto hay que modificarlo y sustituir el uso de bottles de YARP por las variables con los datos procedentes, a través de una subscripción, del topic correspondiente.

Para conseguir enlazar AMOR con los programas de ROS simplemente hay que incluir en el código de YARP los nodos que se van a usar y si van a ser publishers o subscribers, así que se incluyó el código de la figura 3.28 y se modificaron los fragmentos explicados en el párrafo anterior para que YARP pudiese publicar o suscribirse en ROS, tal y como aparece en la figura 3.29.

```
//ROS-------------------------------------------------------------
        yarp::os::Node node1("/yarp/listener");
        yarp::os::Node node2("/yarp/talker");

        yarp::os::Subscriber<Twist> subscriber;

         if (!subscriber.topic("/chatter")) {
            cerr<< "Failed to subscriber to /chatter\n";
            return -1;
        }
        yarp::os::Publisher<Twist> publisher;

         if (!publisher.topic("/posicion2")) {
            cerr<< "Failed to create publisher to /posicion2\n";
            return -1;
        }
//----------------------------------------------------------------
```

Figura 3.28: Nodos declarados

```
-----------------------------------------

   sdata.linear.x=cartesianPositions[0];
   sdata.linear.y=cartesianPositions[1];
   sdata.linear.z=cartesianPositions[2];
   sdata.angular.x=cartesianPositions[3];
   sdata.angular.y=cartesianPositions[4];
   sdata.angular.z=cartesianPositions[5];
   //printf("_dx, _dy, _dz: %f, %f, %f.\n"
   publisher.write(sdata);
-----------------------------------------
```

(a) publisher de posición

```
// receive Cartesian rate
-----------------------------------------------------

        subscriber.read(rdata);
        cartesianRateReceived.at(0)=rdata.linear.x;
        cartesianRateReceived.at(1)=rdata.linear.y;
        cartesianRateReceived.at(2)=rdata.linear.z;
        cartesianRateReceived.at(3)=0;
        cartesianRateReceived.at(4)=0;
        cartesianRateReceived.at(5)=0;
        printf("_dx, _dy, _dz: %f, %f, %f.\n", carte

-----------------------------------------------------
```

(b) subscriber de velocidad

Figura 3.29: primeras modificaciones de cartesian_rate_ AMOR.cpp

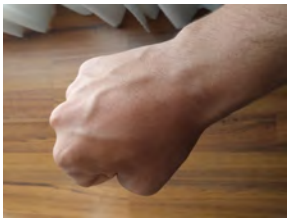
Es importante destacar que en ambos casos el vector utilizado por el programa es de tipo **geometry_msgs**, ver tabla 5.8.

3.2.5. Reconocimiento gestual con Leap Motion

Tras las primeras pruebas de control se comenzó con el diseño de un programa para el reconocimiento gestual mediante Leap Motion antes de empezar con el robot AMOR. Como se ha comentado en la sección 3.1.1, se valoraron tanto la idea de utilizar dos Leap Motion y reservar uno casi exclusivamente para ésto como la de usar sólo uno e intentar concentrar todo el control en una sola mano. Y, de momento, se usó un sólo Leap Motion porque era suficiente para el manejo del proyecto aunque no se descartó el usar dos, en el caso de que hubiera problemas más adelante.

Inicialmente se ideó el uso de tres gestos distintos: uno para abrir la garra, otro para cerrarla y un último gesto para iniciar la ejecución del programa. Pero al final se añadieron dos gestos extras, uno para cambiar el modo de funcionamiento, función que se desarrolló durante la creación de la solución final y que se explicará detalladamente en la siguiente sección, y el otro indica que la mano está en posición neutra.

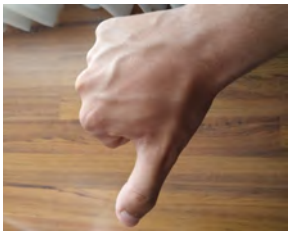
(a) Posición neutra



(b) Cerrar garra

(c) Abrir garra

(d) Inicio



(e) Cambiar modo

Figura 3.30: Gestos disponibles

En cuanto a la diseño de los gestos, se intentó utilizar gestos más o menos intuitivos, por ejemplo, para cerrar la garra se decidió utilizar el puño cerrado, figura 3.30b. Para abrir la garra se iba a utilizar el gesto de la figura 3.30a pero como es el gesto que por defecto se suele utilizar al final se decidió utilizar el de la figura 3.30c. En cuanto al cambio de modo se utilizó el gesto de la figura 3.30e por ser el que me-

jor se detectaba porque, como se explicará en la siguiente sección, se utilizará para lanzar una parada de emergencia. Para iniciar el sistema, figura 3.30d, se escogió un gesto que no pudiera detectarse por casualidad, es decir, que no fuese demasiado natural.

Además se decidió incluir el reconocimiento gestual en un programa externo al que controla al robot para que no interfiriese en sus procesos por lo que se tuvo que crear un topic donde publicar el gesto que está detectando en cada momento.

Cabe destacar que, al estar usando sólo un Leap Motion, los gestos sólo pueden ser formas estáticas en vez de algún tipo de movimiento porque si no entraría en algún conflicto con el sistema de control de las velocidades. El método empleado para la detección es bastante simple porque se basó en el cálculo de la distancia de la punta de cada dedo mediante la ecuación 3.1 y se hicieron pruebas de los valores de cada dedo para clasificar cada gestos en función de los valores de estos. Además, para el gesto que corresponde a la apertura de la garra, figura 3.30c, también se mira la distancia entre la punta del pulgar y del dedo índice para comprobar que están juntos . Por último hay un tipo de gesto que se ha denominado desconocido que significa que el gesto no coincide con ninguno de los definidos. El diagrama del programa se encuentra en la figura 3.31 y la información se publica a través topic de ROS llamado */gestos*.

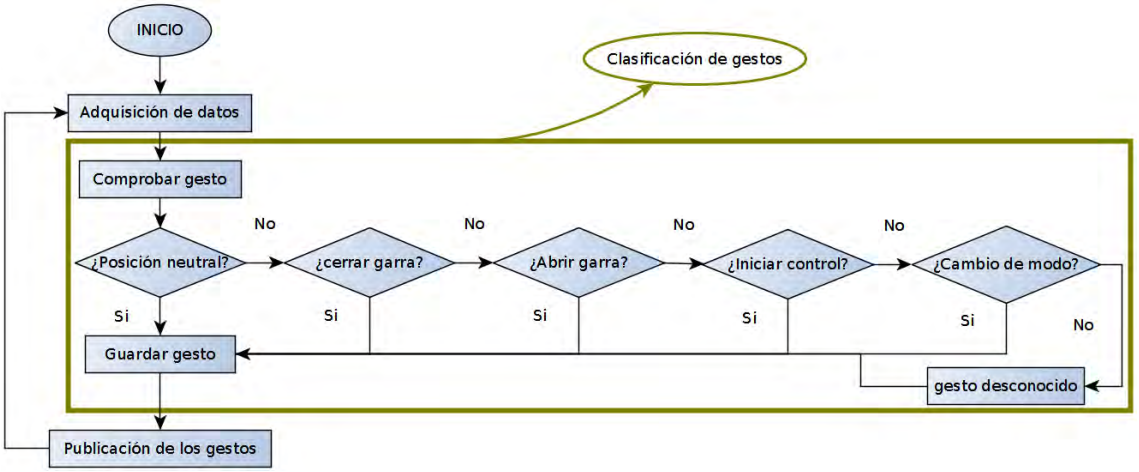


Figura 3.31: Diagrama del reconocimiento de gestos

3.3. Solución final

En cuanto a la solución final va a tener las siguientes características:

- **Leap Motion:** Será capaz de obtener la información de Leap Motion y publicarla en un topic de ROS.
- **Reconocimiento gestual:** Será capaz de diferenciar los cinco gestos que se han mostrado en la figura 3.30 y actuar en consecuencia.
- **Control de AMOR:** Va a tener varios modos de control distintos, uno será el control de posición que se ha usado durante las fases anteriores y luego hay dos modos de control tipo joystick, además de un último modo que envía el robot a una posición inicial. Además llevará incluidos algunos mecanismos de parada.
- **Comunicación con YARP:** Se adecuará el programa de control basado en YARP que utiliza AMOR para poder publicar y suscribirse en los topics de ROS.

En la tabla 3.1 se muestran los distintos topics de ROS utilizados en este proyecto con una breve descripción de la función que tienen.

TOPIC	DESCRIPCIÓN
/leapmotion/data	Guarda la información de Leap Motion
/posicion2	Guarda la posición cartesiana de la garra de AMOR
/gestos	Guarda el nombre del gesto que se detecta
/artpos	Guarda la posición articular de la garra de AMOR
/chatter	Guarda la velocidad a la que se tiene que mover AMOR
/garra	Guarda el nombre del gesto que se detecta
/Mode	Cambia entre modo de velocidades cartesianas y articulares
/return	Habilita la vuelta a una posición inicial
/mano	Indica si Leap Motion detecta la mano o no
/paro	Envía una señal de stop cuando se cambia de modo de control
/cambio	Controla los cambios de modo de control del programa

Tabla 3.1: Topics utilizados

El esquema de las comunicaciones entre los distintos programas se encuentra en la figura 3.32. En este caso se ha decidido no mostrar ROS Master y Yarpserver porque en los esquemas anteriores ya se ha dado la idea de cuál es su función. En la parte inferior derecha hay una leyenda sobre los elementos utilizados.

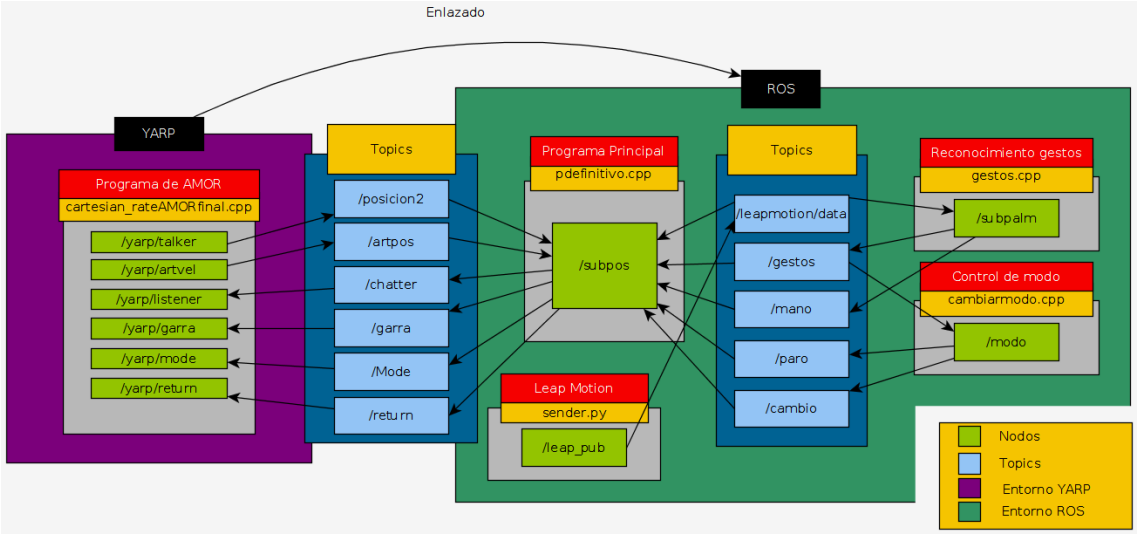


Figura 3.32: Esquema de las conexiones final

Primero, el programa que lleva la parte de Leap Motion, *sender.py*, será el mismo que se ha mostrado en la primera fase, sección 3.2.1, y no se ha modificado en absoluto porque publica todos los datos necesarios. Este programa publicará en el topic */leapmotion/data* toda la información obtenida de Leap Motion utilizando datos de tipo **leapros**, ver tabla 5.7, que viene incluido en el package.

En cuanto al reconocimiento gestual, como se ha comentado en el apartado anterior, se creó un programa a parte, llamado *gestos.cpp* donde el funcionamiento es similar al del apartado anterior, pero se incluyeron nuevas funcionalidades que se pueden ver en la figura 3.33.

Por ejemplo, se introdujo una pregunta antes de empezar para calibrar. Ésta que pide al usuario que extienda la mano para guardar el valor de la distancia desde el centro de la mano a la punta del dedo corazón. Al dividir distancia de los dedos por ese valor se obtiene una 'mano modelo' consiguiendo que el reconocimiento de gestos sea independiente al tamaño de la mano. También se añadió un sistema de parada para frenar al sistema cuando no se está reconociendo ninguna mano. Este

sistema de parada se basa en guardar el dato de la lectura anterior, cotejarlo con el actual y si coinciden significa que no hay ninguna mano porque nunca se obtienen dos valores exactamente iguales, y menos con la mano en suspensión.

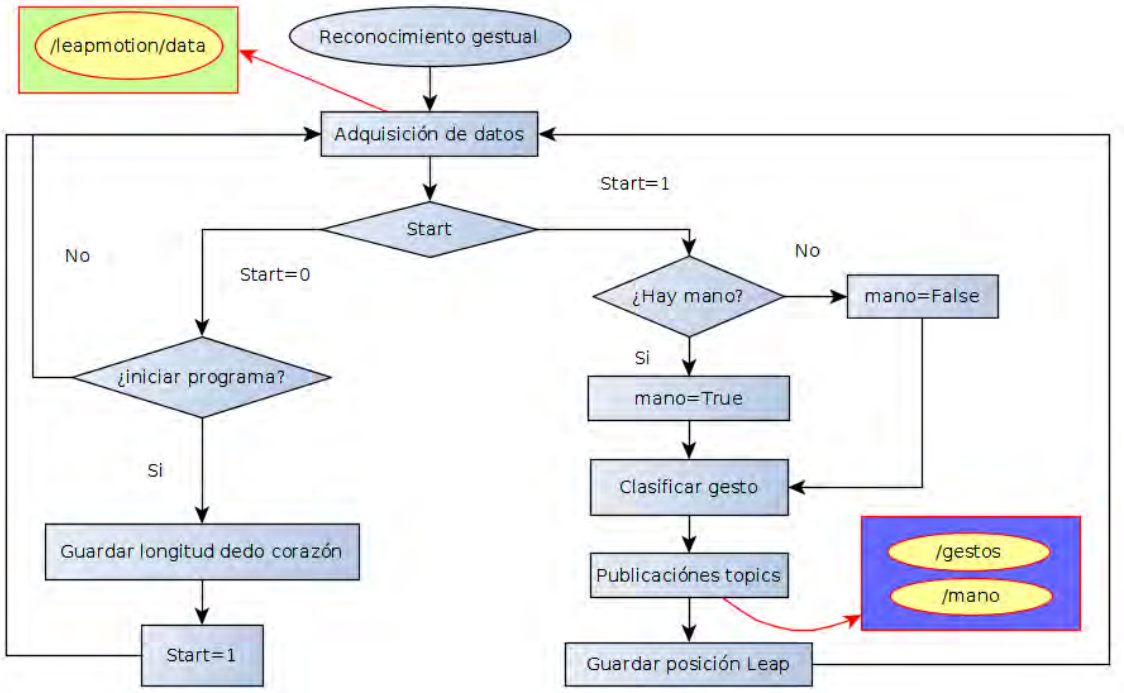


Figura 3.33: Diagrama de la parte de reconocimiento gestual

Este programa publica en el topic */gestos* utilizando datos de tipo **string**, en el topic */mano* utilizando datos de tipo **bool** y está suscrito a */leapmotion/data* que utiliza **leapros**, ver tabla 5.7.

Por otro lado se creó un nuevo programa para controlar el cambio de tipo de control, diagrama de la figura 3.34.

Cuando detecta el gesto de cambio, figura 3.30e, envía una señal de parada del sistema y pide una confirmación del cambio. Si no se quiere cambiar el programa desactiva la señal de parada y vuelve a su posición inicial a esperar el gesto de cambio, pero si la respuesta es afirmativa pregunta a qué modo se quiere cambiar y, a continuación, se publica el nuevo modo y se desactiva la señal de parada. Además es importante señalar que la detección del gesto de cambio es por flanco de subida, es decir, en cada ciclo se compara el gesto nuevo con el anterior y sólo se considera que se ha detectado cuando el nuevo es el de cambio y no coincide con el antiguo,

para evitar que si se mantiene el gesto cambie el modo constantemente.

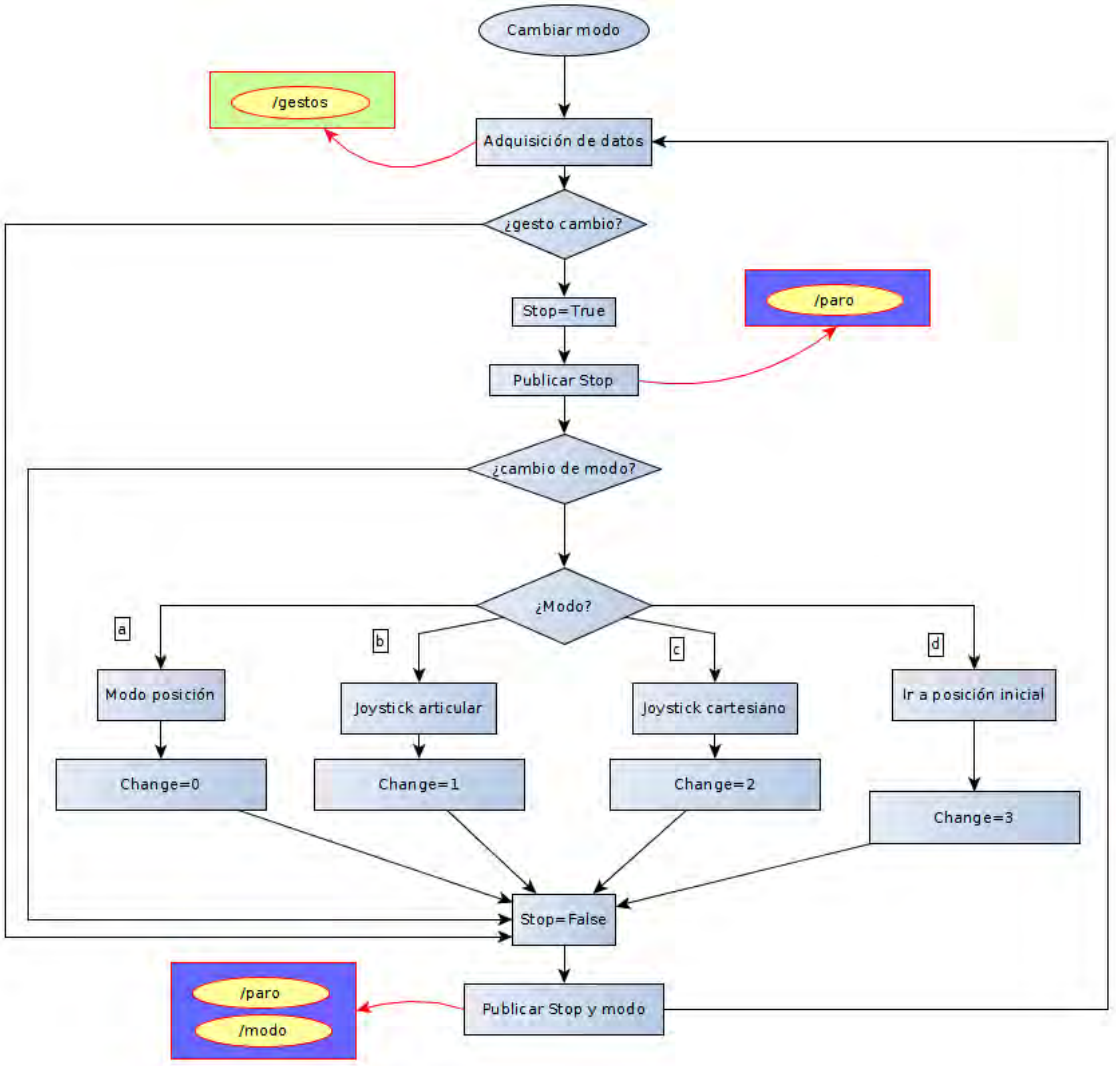


Figura 3.34: Diagrama de la parte del cambio de modo

A continuación está el programa principal llamado pdefinitivo.cp, figura 3.37, que es el núcleo de este trabajo y, como se puede ver en la figura 3.36, el nombre de su nodo es *subpos*. Éste está suscrito a */leapmotion/data*, donde recibe la información de Leap Motion, a */posicion2*, donde obtiene la posición actual cartesiana de la garra de AMOR, a */artpos*, donde recoge las posiciones articulares de AMOR, a */gestos*, donde se está publicando el gesto actual de la mano, a */mano* que indica si se está detectando una mano o no, a */cambio* para cambiar de modo y a */paro* de donde recibe una señal de parada cuando se cambia el modo.

Cada subscripción va enlazada a una función donde se guardan los datos en la variable que se haya elegido. En la mayoría de las subscripciones utilizadas simplemente se realiza esa acción, sin embargo en las funciones **AMORpos** y **palmposMR**, figura 3.35, que corresponden con */posicion2* y */leapmotion/data* respectivamente, su ejecución tiene algunos elementos adicionales.

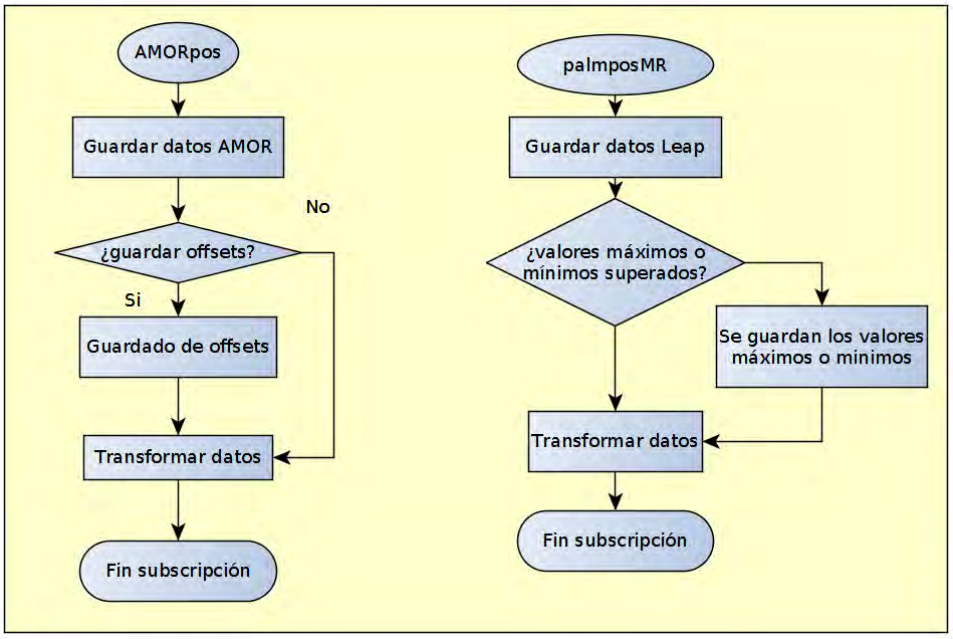


Figura 3.35: Diagrama de AMORpos y palmposMR

En el caso de **AMORpos** se han utilizado unas variables que guardan unos offsets por el mismo motivo que se explicó en la página 58, es decir, para poder ajustar correctamente la transformación al nuevo espacio de coordenadas. En cuanto a **palmposMR** se ha limitado el área de acción para asegurar una zona donde no vaya a haber problemas con el reconocimiento gestual por lo cual, basándonos ligeramente en las características técnicas de la sección 2.3.1, las dimensiones de nuestra interaction box (figura 2.13) son: $x \in (-70, 70), y \in (145, 300)$ y $z \in (-70, 70)$ con centro a 20 cm de Leap, en (0,200,0). Son algo superiores a las de la parte teórica pero se ha comprobado que no hay problemas con el reconocimiento de gestos.

Al mismo tiempo publica en cuatro topics distintos:

- ***/chatter*:** El primer topic es el encargado del control de velocidad y envía datos de tipo **geometry_msgs**, ver tabla 5.8. El programa procesa los datos que le llegan desde el robot y Leap Motion y crea los vectores de velocidad necesarios para su movimiento, que son los que se envían.

- ***/garra*:** Aquí se publican el nombre del gesto que se está utilizando en cada momento y utiliza datos de tipo **string**. Este topic puede parecer innecesario porque es exactamente la misma información que */gestos*, sin embargo su existencia se debe a un problema de comunicación que apareció durante el desarrollo entre ROS y YARP.

- ***/Mode*:** En este topic envía un dato de **bool**, que ha sido llamado *mode* que indica si el sistema está funcionando en modo cartesiano o en modo articular.

- ***/return*:** Por último este topic envía también un dato de tipo **Bool**, que ha sido llamado *pos*, para cambiar entre control articular usado por el joystick articular y el modo de posición inicial.

```
int main(int argc, char **argv){
        ros::init(argc, argv, "subpos");
        ros::NodeHandle n1;
        ros::Subscriber sub = n1.subscribe("posicion2", 1000, &AMORpos);
        ros::NodeHandle n2;
        ros::Subscriber sub2 = n2.subscribe("leapmotion/data", 1000, &palmposMR);
        ros::NodeHandle n3;
        ros::Subscriber sub3 = n3.subscribe("gestos", 1000, &gesture);
        ros::NodeHandle n8;
        ros::Subscriber sub8 = n8.subscribe("artpos", 1000, &ARTposition);
        ros::NodeHandle n4;
        ros::Publisher pub=n4.advertise<geometry_msgs::Twist>("chatter",1000);
        ros::NodeHandle n5;
        ros::Publisher pub2=n5.advertise<std_msgs::String>("garra",1000);
        ros::NodeHandle n6;
        ros::Publisher pub3=n6.advertise<std_msgs::Bool>("Mode",1000);
        ros::NodeHandle n7;
        ros::Publisher pub4=n7.advertise<std_msgs::Bool>("return",1000);
```

Figura 3.36: Declaración del nodo y enlace con los topics correspondientes

Como se ha mencionado anteriormente, el sistema tiene bastantes funciones nuevas respecto a lo que se propuso inicialmente en la sección 3.2.4 y en la figura 3.37 se puede ver el diagrama del funcionamiento completo del programa principal, pdefinitivo.cpp. Además hay dos grupos de bloques, el primero se corresponde con el bucle de inicio mientras que el otro comprende el bucle de ejecución del programa.

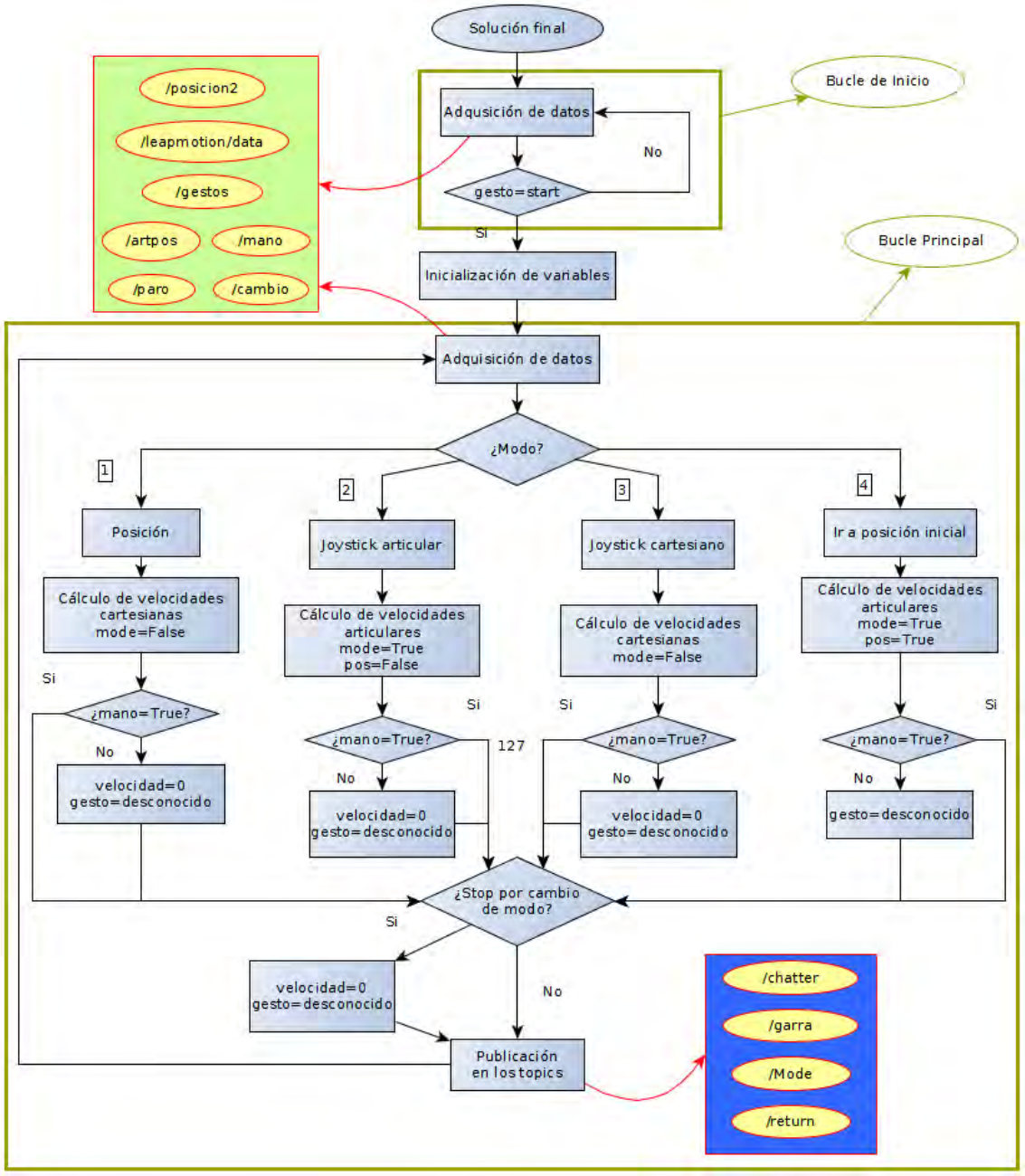


Figura 3.37: Diagrama del programa principal

Las flechas rojas indican una aclaración sobre las comunicaciones que se realizan en los bloques correspondientes.

El bucle de inicio equivale a una espera para que el usuario pueda comenzar cuando decida, en este caso a través del gesto de inicio (figura 3.30d), de tal manera que el programa no empezará hasta que no se realice. Como se puede observar en la figura 3.38, mediante el uso de un bucle while no se deja avanzar al programa

hasta que se detecte ese gesto. En ROS, el programa puede revisar en cada ciclo la información de las subscripciones a través del comando *ros::spinOnce()*. Además *ros::Rate(8).slep()* ajusta la frecuencia del bucle, en este caso 8 Hz.

```
while((ros::ok())&&(gestos.data!="start")){
        ros::Rate(8).sleep();
        ros::spinOnce();
};
```

Figura 3.38: Espera activa de inicio

Inmediatamente después se inicializan las variables de la figura 3.39 antes de comenzar la ejecución del bucle principal. La variable **incio** sirve para permitir la transformación de los datos del robot, **change** es la variable utilizada para la elección de los modos de funcionamiento, **mode** es la variable que indica el uso de cartesianas o articulares, **guardaroffs** permite la actualización de los offset del robot y **joystick** es la posición del punto inicial en los joysticks.

```
inicio=1;
change=3;
mode.data=false;
guardaroffs=1;
//offs joystick
joystick.linear.x=0;
joystick.linear.y=200;
joystick.linear.z=0;
```

Figura 3.39: Inicialización

Tras superar el bucle de inicio el programa llega al bucle de ejecución, así que hay que explicar primero en qué consisten los diferentes modos de control y la razón por la cual se utiliza cada uno.

Primero está el modo de **control de posición** del que se ha hablado varias veces durante este proyecto y el objetivo es el mismo. Cuando se ejecuta por primera vez, tras un cambio de modo, se resetean los offsets guardados para poder utilizar este control en la nueva ubicación.

Sin embargo hay que destacar que se ha añadido una modificación respecto al cálculo numérico de los valores de y evitando cambios bruscos de velocidad la velocidad para estabilizarlos y evitar cambios demasiado bruscos. Esta vez no se basa

en una progresión aritmética proporcional sino que se rige por las ecuaciones (3.5), (3.6) y (3.7) para intentar obtener una velocidad relativamente constante en vez de valores demasiado pequeños como cuando los puntos relativos de la mano y la garra estaban bastante cerca o valores demasiado elevados en el caso contrario. Las constantes K se utilizan para ajustar el correcto funcionamiento de estas ecuaciones y el signo depende del valor de la distancia, es decir, si $(x_{AMOR} - x_{mano})$ es negativo el signo también será negativo y viceversa.

$$v_{x_{amor}} = \pm K_v \times \sqrt[10]{K_x \times (x_{AMOR} - x_{mano})^2} \tag{3.5}$$

$$v_{y_{amor}} = \pm K_v \times \sqrt[10]{K_y \times (y_{AMOR} - z_{mano})^2} \tag{3.6}$$

$$v_{z_{amor}} = \pm K_v \times \sqrt[10]{K_z \times (z_{AMOR} - y_{mano})^2} \tag{3.7}$$

En la figura 3.40 se puede observar la diferencia entre el uso de una progresión aritmética y la ecuación que se está utilizando consiguiendo valores de velocidad relativamente estables. La gráfica se ha realizado con el programa llamado Gnuplot que está disponible de forma gratuita en Ubuntu.

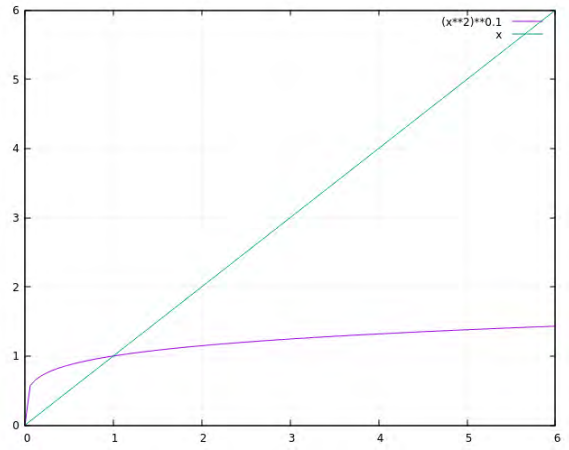


Figura 3.40: Gráfica de comparación entre ecuaciones

La linea azul corresponde con la ecuación $y = \sqrt{(x_2 - x_1)^2}$ que se estaba utilizando y la morada a la ecuación que se está utilizando.

Por otro lado, las velocidades que se utilizan son respecto a la garra así que hay que transformar las velocidades al sistema de referencia del robot.

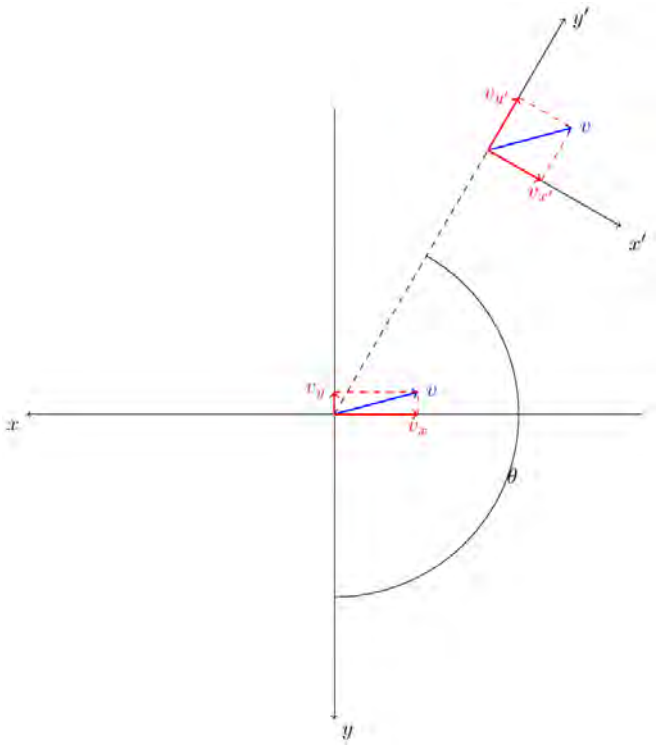


Figura 3.41: Transformación de la velocidad

En la figura 3.41 se puede observar visualmente la transformación que hay que realizar. Como el eje **z** no varía en la figura sólo sale representado el plano xy para que se aprecie mejor. El vector v está representado en ambos sistemas con la misma dirección y en cada sistema se observan sus proyecciones correspondientes. La ecuación (3.8) muestra la matriz de transformación para pasar de un sistema de referencia a otro y es la que se empleará en el programa, aunque sólo es necesaria para las velocidades cartesianas.

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sen(\theta) & 0 \\ sen(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} \tag{3.8}$$

Volviendo con el control de posición, el funcionamiento se basa en la idea de la figura 3.2b. El cálculo de las velocidades, figura 3.42, se realiza mediante la función *getvelocity* que equivale a $v = \pm \sqrt[10]{d^2}$ que es la fórmula genérica de las ecuaciones (3.5), (3.6) y (3.7) siendo $d = x_2 - x_1$. La función *getlineardistance* utiliza la ecuación $y = \sqrt{(x_2 - x_1)^2}$ para calcular distancias

```
if (getlineardistance(data.palmpos.x,yarpmsg.linear.x)<20){
        vel.linear.x=0;
        distancia.linear.x=0;
}else{
        distancia.linear.x=(data.palmpos.x-yarpmsg.linear.x)/50;
        vel.linear.x=getvelocity(distancia.linear.x)/6;
}
```

Figura 3.42: Fragmento del código para el cálculo de las velocidades

Por último se necesita incluir el código de la transformación de la ecuación (3.8), figura 3.43, donde además también se puede apreciar el código de parada en el caso de que no se detecte ninguna mano mediante el borrado de los valores de velocidad y gestos.

```
if (mano.data==false){
vel.linear.x=0;vel.linear.y=0;vel.linear.z=0;
gestos.data="desconocido";
}
vel1=vel;
vel.linear.x=-(vel1.linear.x*cos(yarpmsg.angular.x)-vel1.linear.y*sin(yarpmsg.angular.x));
vel.linear.y=-(vel1.linear.x*sin(yarpmsg.angular.x)+vel1.linear.y*cos(yarpmsg.angular.x));
```

Figura 3.43: Transformación de las velocidades

A continuación, el segundo modo es el **joystick articular**. La razón por la que se ha incluido es el escaso rango de acción que tiene el método de control de posición, limitado por el área de interacción de Leap Motion. Así que, para intentar solucionar este problema, se añadió esta nueva función que permite rotar el eje de la base, articulación A1 (figura 3.44), del robot y el control de la articulación A5 (figura 3.45).

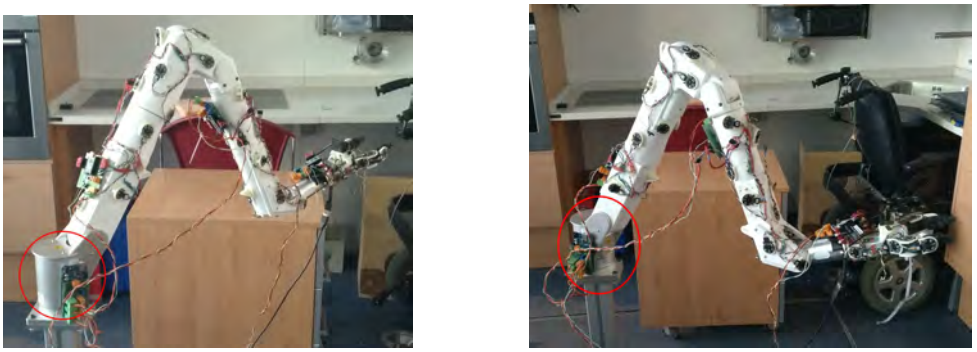


Figura 3.44: Articulación A1

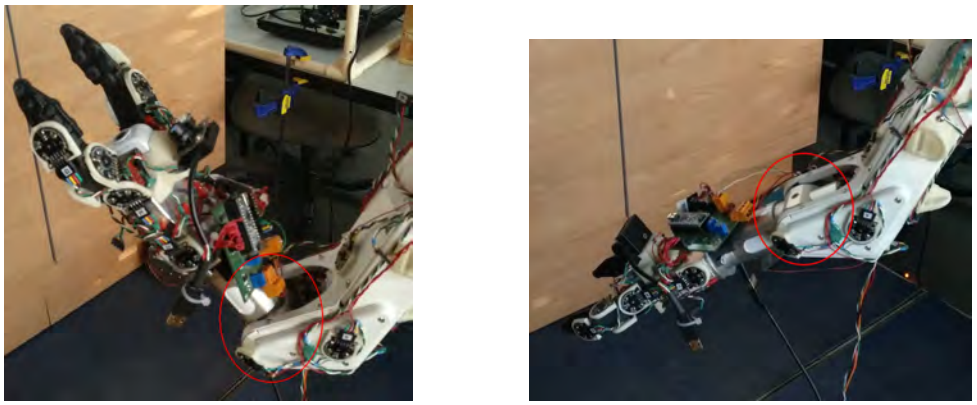


Figura 3.45: Articulación A5

Utilizando la idea de la figura 3.2a, el funcionamiento consiste en establecer un punto situado en el espacio de Leap Motion como punto inicial, en este caso es (0,200,0), que se guarda en la variable joystick de tipo **geometry_msgs** (ver tabla 5.8) y se crea un vector utilizando el punto central de la mano como punto final. En este caso, sólo se usan dos ejes, el eje **x** controla la velocidad de giro de la articulación A1 de la figura 2.16 mientras que el eje **z** controla la articulación A5 de la misma figura. Para el cálculo de velocidades y la parada se usan los mismos métodos que en el modo anterior.

El siguiente modo de control es un **joystick cartesiano**. La idea es ofrecer al usuario múltiples formas de control para que escoja la que más le guste por lo que al final se optó por incluir los dos métodos propuestos en la página 41. Su funcionamiento es el mismo que en el modo anterior salvo por dos diferencias: ahora se está utilizando movimiento cartesiano por lo que el valor de la variable **mode** será *false* y esta vez sí se utilizarán los tres ejes de coordenadas. La principal ventaja que tiene este método frente al de control de posición reside en que no está delimitado por ningún rango máximo de acción.

Las ecuaciones para el cálculo de las velocidades son las mismas que se han utilizado anteriormente y, al igual que en el primer modo, se utiliza el espacio cartesiano por lo hay que transformar el valor de las velocidades utilizando la ecuación (3.8).

Figura 3.46: Posición inicial de AMOR

El último modo de control corresponde con el envío del robot a una **posición inicial** previamente establecida (figura 3.46). Se utiliza un movimiento articular por lo que el valor de **mode** será **true** y es importante destacar que se ha diseñado de tal forma que la posición inicial sea solidaria con la base, es decir, en realidad no es una posición concreta en el espacio, son un conjunto de puntos que forman una circunferencia alrededor de la articulación A1. En la figura 3.44 se puede observar la posición inicial con una orientación diferente de la articulación de la base.

La fórmula utilizada para el cálculo de velocidades es la misma que en los modos anteriores, sin embargo, en este caso se calcularán seis valores de velocidad, uno por cada articulación, despreciando la base (articulación A1), a partir de una posición articular inicial escogida y la posición de esas articulaciones del robot. Cuando se alcanza la posición, el robot se queda estático hasta que se cambie de modo. Además, en este modo AMOR no frena cuando se quita la mano, sólo se borran los gestos, porque, como Leap Motion no influye en el control de la velocidad en esta parte, no tendría sentido mantener esa condición.

Por último existe otro mecanismo de parada, que esta vez funciona en todos los modos de control, y tiene que ver con el cambio de modo. Se ha decidido introducir este stop para que, cuando ejecute el gesto de cambio de modo, el robot se pare para permitir al usuario el control del ordenador para elegir el siguiente modo. Además, aunque no sea su función principal, se podría usar este mecanismo como parada de emergencia porque es el único que afecta a todos los modos de control. Y otra cosa a tener en cuenta es que cada vez que se cambie de un modo al de control de posición se reinicie el guardado de offsets para ajustar la nueva ubicación, como se ha comentado anteriormente.

Para terminar queda el programa de AMOR, basado en YARP, que se llama *catesian_rate_AMORfinal.cpp* y donde hay algunos cambios respecto a lo que se mostró en la sección 3.2.4. Este programa tiene seis nodos enlazados con seis topics, declarados en el código en la figura 3.47 y que se pueden ver claramente en el esquema de comunicaciones de la figura 3.32.

```
//ROS-----------------------------------------------------------
        yarp::os::Node node1("/yarp/listener");
        yarp::os::Subscriber<Twist> subscriber;

         if (!subscriber.topic("/chatter")) {
            cerr<< "Failed to subscriber to /chatter\n";
            return -1;
        }
        yarp::os::Node node2("/yarp/talker");

        yarp::os::Publisher<Twist> publisher;

         if (!publisher.topic("/posicion2")) {
            cerr<< "Failed to create publisher to /posicion2\n";
            return -1;
        }
        yarp::os::Node node6("/yarp/artvel");
        yarp::os::Publisher<Twist> publisher2;

         if (!publisher2.topic("/artpos")) {
            cerr<< "Failed to create publisher to /artpos\n";
            return -1;
        }
        yarp::os::Node node3("/yarp/garra");
        yarp::os::Subscriber<String> subscriber2;

         if (!subscriber2.topic("/garra")) {
            cerr<< "Failed to subscriber to /garra\n";
            return -1;
        }
        yarp::os::Node node4("/yarp/mode");
        yarp::os::Subscriber<Bool> subscriber3;

         if (!subscriber3.topic("/Mode")) {
            cerr<< "Failed to subscriber to /Mode\n";
            return -1;
        }
        yarp::os::Node node5("/yarp/return");
        yarp::os::Subscriber<Bool> subscriber4;

         if (!subscriber4.topic("/return")) {
            cerr<< "Failed to subscriber to /return\n";
            return -1;
        }

//-------------------------------------------------------------
```

Figura 3.47: Topics enlazados con AMOR

El programa publica en */posicion2* la posición cartesiana de la garra de AMOR y en */artpos* la posición de las articulaciones, utilizando en ambas publicaciones datos de tipo **geometry_msgs** (ver tabla 5.8), mientras que, a la vez, está subscrito a los 4 topics que se publican desde el programa principal y que están explicados en la página 69. En cuanto a la programación de esta parte, hay que añadir tres funciones

nuevas que no estaban en la sección 3.2.4. Éstas son la apertura y cierre de la garra y el cambio de modo entre movimiento cartesiano y articular.

El cierre y la apertura de la garra se controla mediante el código de la figura 3.48. El programa recibe un dato de tipo **string** a través de topic */garra* con el gesto que se esté detectando. Como se puede ver en la figura, el código original utiliza dos variables que son *button1* y *button2* para abrir y cerrar la garra respectivamente. Por lo tanto, cuando se detecta el gesto abrir, *button1* se activa mientras que se desactiva la otra, si se detecta cerrar es la variable *button2* la que se activa y *button1* la que se desactiva. Si se detecta cualquier gesto que no corresponda a ninguno de los dos, las dos variables se desactivan y se frena la apertura o cierre de la garra.

```
//-------------------------------------------------------------------------------
//ROS comunication
                        subscriber2.read(garradata);
                        cout<<"Recieved:"<<garradata.data<<endl;
                        if (garradata.data=="abrir"){
                                button1=1;
                                button2=0;
                        }
                        if (garradata.data=="cerrar"){
                                button1=0;
                                button2=1;
                        }
                        if (garradata.data!="cerrar" && garradata.data!="abrir"){
                                button1=0;
                                button2=0;
                        }
//-------------------------------------------------------------------------------
```

Figura 3.48: Control de la garra de AMOR

En un principio el topic */garra* no se iba a utilizar porque se había pensado aprovechar */gestos* que guarda exactamente los mismos datos. Sin embargo cuando se realizaron las primeras pruebas de este programa surgieron algunos problemas de comunicación que se consiguieron resolver juntando todos los publishers de los topic utilizados en YARP en el programa principal. La hipótesis principal del problema es que, al publicar desde dos programas distintos, el envío de datos a los topics no estaba sincronizado dado ROS utiliza comunicación asíncrona, es decir, siempre publica la información correspondiente según la recibe. A pesar de que no hay ningún problema cuando se hace así en ROS, suponemos que ésto impedía que funcionara correctamente en YARP porque, una vez modificado, el sistema funcionaba sin ninguna dificultad.

Otra nueva función del programa es el cambio de modo para utilizar velocidades

articulares o cartesianas y se controla mediante la variable **mode** de tipo **Bool** que llega a través del topic */gestos*. Si es falsa, el robot utilizará el modo de control cartesiano pero, si es verdadera, se usará el modo articular.

```
                        // receive Cartesian rate
//------------------------------------------------------------------------
//ROS comunication
                        subscriber.read(rdata);
                        subscriber3.read(mode);
                        subscriber4.read(pose);
                        //cout<<"data:"<<rdata.linear.x<<", "<<rdata.linear.
                        if (mode.data==false){
                                cartesianRateReceived.at(0)=rdata.linear.x;
                                cartesianRateReceived.at(1)=rdata.linear.y;
                                cartesianRateReceived.at(2)=rdata.linear.z;
                                cartesianRateReceived.at(3)=0;
                                cartesianRateReceived.at(4)=0;
                                cartesianRateReceived.at(5)=0;
                                printf("_dx, _dy, _dz: %f, %f, %f.\n", carte

//------------------------------------------------------------------------
```

Figura 3.49: Control cartesiano de AMOR

Como se puede ver en la figura 3.49, al principio se recibe la información de las subscripciones y, en función del mode, se escoge el tipo de control. En la figura se encuentra también el control cartesiano que es igual al que se explicó al final de la sección 3.2.4 donde, en principio, solo se va a utilizar la velocidad lineal porque las velocidades angulares cartesianas no funcionaban demasiado bien con la API de AMOR que está instalada actualmente.

```
}else{
        if(pose.data==false){
        g_jointVelocity[0]=rdata.linear.x;
        g_jointVelocity[1]=0;
        g_jointVelocity[2]=0;
        g_jointVelocity[3]=0;
        g_jointVelocity[4]=0;
        g_jointVelocity[5]=rdata.linear.z;
        g_jointVelocity[6]=0;
        }else{
        g_jointVelocity[0]=0;
        g_jointVelocity[1]=rdata.linear.x;
        g_jointVelocity[2]=rdata.linear.y;
        g_jointVelocity[3]=rdata.linear.z;
        g_jointVelocity[4]=rdata.angular.x;
        g_jointVelocity[5]=rdata.angular.y;
        g_jointVelocity[6]=rdata.angular.z;
        }

        cout<<"vel="<<g_jointVelocity[0]<<", "<<g_jointVelocity[1]<<", "<<g_
ndl;

        if(amor_set_velocities(g_hRobot,g_jointVelocity) != AMOR_SUCCESS) {
                HandleError();
        }
}
```

Figura 3.50: Control articular de AMOR

Por otro lado, en la figura 3.50 que muestra el control articular, AMOR utiliza un vector de 7 elementos debido a que tiene 7 grados de libertad. En el modo del joystick sólo se están utilizando el eje A1 y el A5, sin embargo en el modo para ir a una posición inicial se utilizan todas las articulaciones menos A1. Como no utilizan la misma estructura de datos se incluyó la variable **pose** de tipo **Bool** ligada al topic */return* que permite utilizar la estructura adecuada en cada caso.

En cuanto a la publicación de la posición de la garra de de AMOR se realiza exactamente igual a como se hizo en la figura 3.29a en la sección 3.2.4 donde se guarda en una variable llamada sdata de tipo **geometry_msgs**, ver tabla 5.8, aunque en este caso también se publica las posiciones de las articulaciones utilizando el mismo mecanismo.

Para la instalación hay que seguir los pasos descritos en github en el enlace *https://github.com/gendibal784/AMORcontrol.git* donde se encuentran los packages utilizados y en el Apéndice, página 101, se encuentra una guía para la secuencia de ejecución de cada uno de los programas utilizados. Los códigos completos utilizados se encuentran dentro de los packages del enlace anteriormente mencionado.

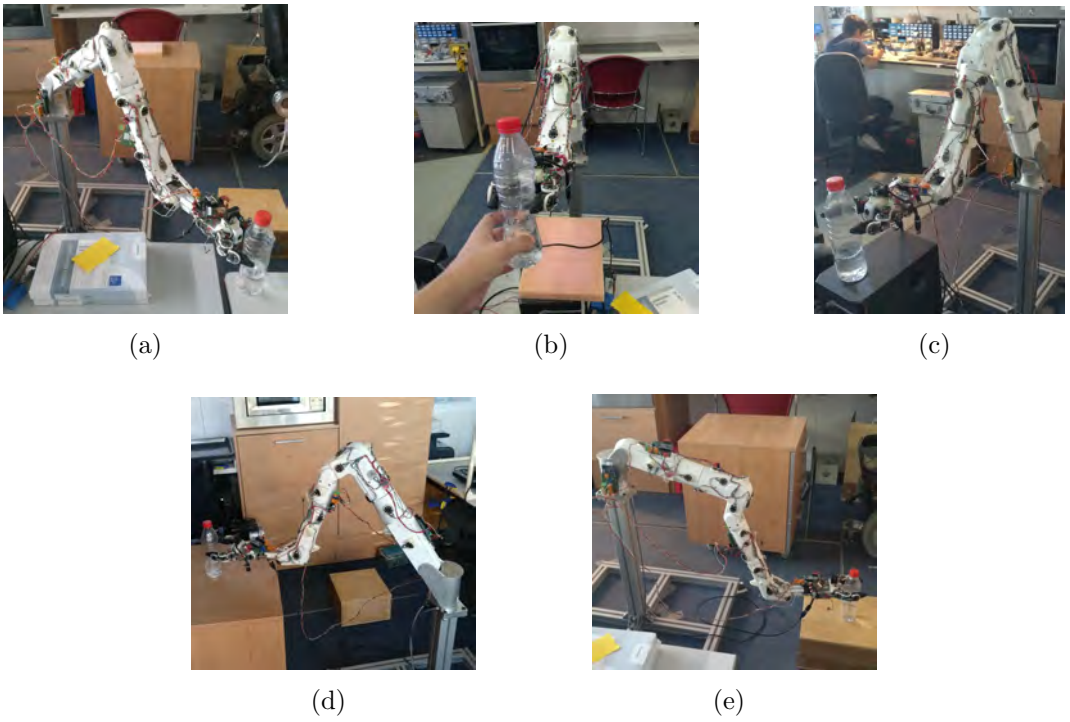
Capítulo 4

Análisis de resultados

Índice

4.1. Pruebas realizadas

En esta sección se van a exponer las pruebas que se han realizado. Los primeros experimentos se han centrado en el agarre de un objeto, figura 4.1, en distintos lugares intercalando tanto posiciones elevadas, a media altura o bajas para trasladarla hacia una nueva ubicación o para interactuar con el usuario. Al utilizar una sola mano para controlar AMOR, la otra queda libre y puede realizar tareas como recoger el objeto que está agarrando el robot, figura 4.1b.



(a) (b) (c)

(d) (e)

Figura 4.1: Pruebas de alcance

A continuación se realizaron otras pruebas con un enfoque más práctico sobre la robótica asistencial.

Una de las pruebas se basó en el manejo de un microondas, en concreto de su apertura y cierre, figura 4.2. Partiendo desde una posición inicial, se trasladó el robot hasta el botón de apertura de la puerta y se pulsó, figura 4.2a. Una vez abierto comenzó la operación de cierre de la puerta, figura 4.2b. También se simuló la recogida e introducción de objetos en el microondas.

(a) Apertura (b) Cierre

Figura 4.2: Manejo del microondas

Uno de los proyectos del departamento de Robótica Asistencial es la integración del brazo robótico AMOR en una silla de ruedas así que, con la colaboración de un compañero, se probó el manejo con Leap Motion de AMOR desde una silla de ruedas posicionando AMOR en una ubicación similar a la que tendrá en el futuro, simulando una situación real donde el paciente pueda mover el brazo robótico desde la silla de ruedas.

Figura 4.3: Silla de ruedas

Como se puede ver en la figura 4.3, el paciente controlaría AMOR con la mano derecha. El dispositivo Leap Motion se encuentra a 20 cm del reposabrazos que coincide con el centro del área de interacción que se ha establecido previamente. El cambio de modo se introdujo desde el ordenador que estaba conectado a Leap y a AMOR pero se ha planteado como futuras mejoras la creación de una interfaz para algún tipo de dispositivo móvil para no tener que hacerlo desde el ordenador. La prueba consistió en coger la botella naranja situada en la parte derecha de la foto, acercarla a la silla de ruedas y que el paciente pudiese cogerla con una mano.

Además es importante destacar que el compañero que realizó esta prueba no tenía ningún conocimiento previo sobre el sistema de control y tras una breve explicación de los gestos a utilizar y cómo funcionaba cada modo pudo realizar el ejercicio sin problemas.

4.2. Análisis crítico de los resultado

Tras la ejecución de los ejercicios probando y ajustando el control de AMOR se han evaluado los puntos fuertes y los débiles del programa.

En primer lugar el **control de posición** es cómodo porque te permite situar la mano en la posición en el espacio donde quieres situar el robot y solo tienes que esperar a que la alcance. Sin embargo el rango de acción es demasiado reducido por dos motivos: el reconocimiento de gestos y la precisión que quieres tener. Al tener que realizar el reconocimiento gestual con la misma mano que controla el movimiento del robot, hay zonas donde Leap Motion es capaz de monitorizar la posición de la mano pero no reconoce apenas los gestos que se realizan, que es la razón por la que se ha incluido el área de interacción pero que influye negativamente en este modo. Por otro lado, si quieres realizar movimientos con cierta precisión no puedes simplemente incrementar el área de acción del robot en base valores proporcionales mas elevados porque el área de control de Leap Motion sigue siendo la misma. Asi que, tras haberse probado, funciona bien pero como se ha comentado el rango de acción es bastante más reducido de lo esperado limitando su utilidad sólo a posibles tareas de aproximación final.

En segundo lugar se encuentra el modo **joystick articular** donde se controlan las articulaciones A1 y A5 (figura 2.16). Su uso es muy intuitivo porque el giro de las articulaciones es solidario al movimiento que realiza la mano, por ejemplo, si se mueve la mano a la derecha, A1 gira en ese sentido de una forma bastante evidente lo que te permite comprobar visualmente lo que estás haciendo y al igual sucede con A5 y la subida o bajada de la mano. El rango de acción que otorga al robot es inmenso porque permite acceder a los 360º en torno a su eje principal, como se puede observar en las pruebas de alcance de la figura 4.1, y además es el único modo donde realmente se puede modificar la orientación de la garra, ya sea mediante la rotación del eje principal como con el control de pitch de la garra que otorga el eje A5. Esto lo que le convierte en una de las claves del sistema control de AMOR.

En tercer lugar, el modo **joystick cartesiano** dio muy buenos resultados porque cubrió perfectamente las carencias que tenía el control de posición. Es muy útil para moverse durante tramos largos y aporta un gran rango de acción. Durante las pruebas se pudo apreciar una gran sinergia entre este modo y el joystick articular porque

mientras que uno reorientaba el robot, el otro se encargaba de mover el robot hasta el objetivo correspondiente y finalmente éstos modos se convirtieron en el núcleo de control del sistema.

Por último el modo para **ir a una posición inicial** es un tanto atípico porque no necesita casi ningún tipo de control, sólo para abrir y cerrar la garra. Realmente fue bastante más útil de lo que aparenta porque, al ofrecer una posición de inicio solidaria con el giro del eje principal, permite utilizarlo tanto para empezar un movimiento como de punto intermedio de un trayecto, es decir, el robot recoge un objeto de la mesa, se le ordena ir a la posición inicial y desde ahí es más fácil reorientarlo o moverlo hacia otra ubicación.

En la figura 4.4 se muestra el funcionamiento ideal del sistema. Primero se mueve hacia la posición inicial del sistema para partir desde una ubicación conocida y a continuación el grueso del movimiento se realiza con los joysticks hasta que, si es necesario, se utiliza el control de posición para la maniobra de aproximamiento.

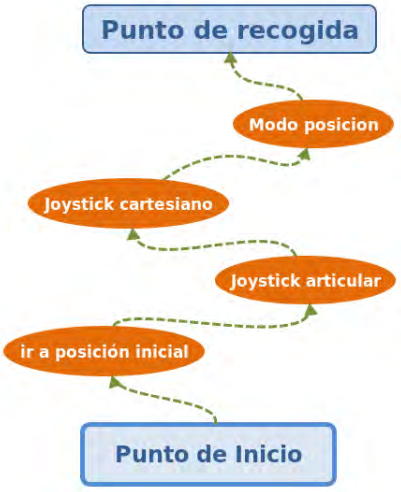


Figura 4.4: Funcionamiento ideal del modo de control

Respecto al tema de seguridad, es decir, los mecanismos de parada del robot son bastante accesibles y sencillos. El primero, el que detiene el robot si no se detecta ninguna mano, es muy intuitivo para la gente que no está familiarizada con el sistema. Por otro lado, la parada del robot cuando se esté cambiando de modo permite al usuario, este caso, utilizar tranquilamente el ordenador para poder seleccionar el nuevo modo. Cabe destacar que se podría emplear como mecanismo de emergencia

para frenar al robot porque es independiente del modo que se esté utilizando.

El cálculo de velocidades, según la explicación de página 72, el valor es prácticamente constante y se podría valorar el uso de un valor constante que sustituiría ese cálculo. Sin embargo, el uso de las ecuaciones de la página 72 permite ajustar la pendiente que se desee porque depende del grado de la raíz, es decir, si disminuye el grado se incrementaría la pendiente de la gráfica y viceversa. Esto facilita los ajustes del programa.

El resultado del reconocimiento gestual fue, en un principio, menos satisfactorio de lo esperado. Si sitúas la mano en posiciones demasiado cercanas o escoradas muchas veces no se detectaba el gesto porque Leap Motion era incapaz de recoger correctamente los datos de la mano lo que, por ejemplo, dificulta enormemente la apertura de la garra, figura 3.30c, que suele ser el más complicado al tener que reconocer la interacción entre el pulgar y el dedo índice. Esto se debe a que el programa utilizado es bastante sencillo porque no se ha incorporado ningún sistema de aprendizaje o el uso de vectores que relacionen los dedos adyacentes como en algunos de los trabajos consultados. Sin embargo, tras haber reducido el rango de acción de Leap Motion los resultados mejoraron considerablemente, a costa del control de posición, al utilizar una zona similar al área de interacción de la figura 2.13, donde Leap Motion es mucho más preciso. Además, el sistema de la 'mano modelo' permite detectar correctamente manos de cualquier tamaño aunque cuando se guarda el valor del dedo central para utilizarlo de referencia a veces es incorrecto por lo que se recomienda utilizar el visualizador de Leap Motion, figura 3.6, para asegurarte de que te está correctamente la mano antes de guardar dicho valor.

Por último, en la sección anterior se comenta que, al emplear una sola mano para el control del robot, ésto permite al usuario utilizar la otra mano para interactuar con el robot. Sin embargo hay un inconveniente con la implementación del reconocimiento gestual y el control de velocidades en una misma mano y es que a veces, cuando se realiza el gesto correspondiente, la mano no se mantiene en la misma posición y puede derivar en pequeños movimientos involuntarios no deseados. Muchas veces el margen de seguridad para empezar el movimiento suele evitarlos pero durante los experimentos sucedió un par de veces, aunque no implicó ningún problema importante, simplemente se reajustaba la posición del robot y estaba solucionado.

Capítulo 5

Conclusiones

Índice

5.1. Conclusiones

El objetivo de este proyecto era el diseño de un sistema de control para el brazo robótico AMOR a través de Leap Motion con la finalidad de conseguir un control sencillo y accesible para personas con discapacidad o limitaciones locomotrices para poder ayudarlas en su día a día y aumentar su nivel de independencia.

El sistema de control diseñado funciona satisfactoriamente como se puede ver en la sección 4.1 donde se han probado diversas tareas diarias como recoger objetos o abrir un microondas. Además, el compañero que realizó la prueba de la silla de ruedas, figura 4.3, no tenía conocimientos sobre el control y tras una breve explicación pudo realizar la prueba correctamente aunque también hay que tener en cuenta que está familiarizado con el robot y su manejo lo que facilitó el control. Por lo tanto, a pesar del los buenos resultados, en un futuro se debería probar con personas ajenas a este área de trabajo para conocer su opinión sobre este sistema de control, si les parece cómodo, si se cansan, si es intuitivo, etc.

Otro aspecto a tener en cuenta en esa misma prueba es que el usuario destacó que daba un poco de impresión cuando se movía el robot hacia la silla de rueda debido a la proximidad, asi que se debería incluir algún tipo de mecanismo de seguridad para evitar posibles colisiones que no dependan del usuario, como los sistemas de frenado que están implementados.

En un principio se planteó basar todo el movimiento en el modo posición. No obstante, a la hora de la práctica el rango de acción es demasiado reducido y su uso terminó siendo bastante residual. En contraposición, los controles de joystick, tanto el articular como el cartesiano, demostraron muy buena sinergia y, al final, el grueso del control se realizaba con estos modos. Durante las pruebas fueron los controles más cómodos pero, como se ha mencionado antes, se debería probar con gente que no esté familiarizada con éste área.

En cuanto al reconocimiento gestual, la introducción de la 'mano modelo' mejoró considerablemente el resultado de la detección de gestos permitiendo reconocer manos de diferentes tamaños para así poder abarcar un mayor número de usuarios. Sin embargo, como se ha mencionado al principio, este sistema implica una movilidad mínima en los dedos para poder realizar los gestos así que, teniendo en cuenta que

está enfocado a la robótica industrial, se debería diseñar algún otro tipo de control alternativo como el control de voz para incrementar el número de pacientes que podrían utilizarlo.

5.2. Posibles mejoras y ampliaciones

En primer lugar, respecto a las mejoras del proyecto propuesto, se centrarían en la mejora del sistema de reconocimiento gestual para poder reconocer más gestos con una mayor precisión lo que añadiría más funcionalidades al robot y optimizar los programas propuestos, mejorando el código utilizado. Además, debido a que en la API de AMOR no funcionaba las velocidades de orientación en cartesianas, no se han podido utilizar así que otra opción sería mejorar la API de AMOR para poder utilizar estos vectores y así poder controlar la orientación de la garra. Por otro lado se podría desarrollar un sistema para transformar las velocidades cartesianas en articulares para así evitar la necesidad del control de la orientación en cartesianas.

Como se ha mencionado al final de la sección anterior, se debería introducir algún tipo de control sin gestos para gente con movilidad reducida en los dedos. Por ejemplo, la implementación de un sistema de reconocimiento de voz facilitaría enormemente el control del sistema a una persona con esos problemas. También seria interesante introducir algún mecanismo de seguridad para evitar colisiones con los futuros usuarios.

En cuanto a futuras ampliaciones, una de las más interesantes se basaría en el desarrollo de una interfaz virtual que pudiese ser controlada a través del ordenador o de algún dispositivo móvil para mejorar la accesibilidad al manejo del programa. Por ejemplo, se podría incluir una opción para ajustar los parámetros del programa, es decir, poder modificar el rango de acción que se utiliza en el modo de control de posición, aumentar o disminuir la velocidad utilizada o, incluso, tener algún sistema que diera la posibilidad al usuario de diseñar los gestos con los que se controla el programa para que escogiese los de mayor comodidad. Además también podría organizar mejor los datos y presentarlos en la interfaz de una manera clara y concisa, en vez de utilizar la terminal de Linux.

Otra posible ampliación sería la adaptación de la API de AMOR a un programa basado en ROS para no tener que utilizar ROS y YARP al mismo tiempo porque, a pesar de haber mecanismos de comunicación, el diseño de la comunicación entre ambas plataformas fue un verdadero rompecabezas. Hay veces que funcionaba con retardo o, al ser dos plataformas distintas ,la terminología de su lenguaje es distinto así que la mejor solución para evitar estos contratiempos sería implementarlo todo en una misma plataforma.

5.3. Entorno socio económico

5.3.1. Marco Regulador

El ámbito jurídico de la robótica todavía no tiene una legislación clara pero algunos países, como EE UU, Japón, China o Corea del Sur, ya se están evaluando adoptar medidas reguladoras en el área de la robótica y la inteligencia artificial, reflexionando sobre cambios legislativos para enfrentar las nuevas aplicaciones de dichas tecnologías; contrastando los riesgos relacionados con la vida y seguridad humana, la intimidad, la integridad, la dignidad, la autonomía y la propiedad de los datos.

En el ámbito europeo, el 20 de enero del 2015, la Comisión Europea creó un grupo de trabajo sobre las cuestiones jurídicas relacionadas con la evolución de la robótica y la inteligencia artificial. Y el 31 de mayo de 2016, se presentó un informe [19] concretando una primera aproximación de normas de Derecho Civil aprobadas por los miembros del Comité del Parlamento Europeo para Asuntos Legales, con una moción para garantizar estatus legal a los robots, a los que se les otorga la condición de "personas electrónicas". La propuesta dispone que "los robots autónomos más sofisticados podrían recibir el estatus de persona electrónica, con derechos y obligaciones específicos", incluyendo la de subsanar los daños que causen.

En el informe [19] se realizó un estudio exhaustivo ámbito de la robótica y la inteligencia artificial y se expusieron una serie de recomendaciones entre las que se encuentra la propuesta que se ha mencionado en párrafo anterior y una serie de códigos éticos y deontológicos que los investigadores en el campo de la robótica deberían comprometerse a adoptar de manera estricta.

Actualmente la normativa vigente más relevante es la realizada por el norma ISO 10218:1992. Es relativamente reciente, pues data del año 1992 y, a grandes rasgos, contiene la siguiente información: una sección sobre el análisis de la seguridad, la definición de riesgos y la identificación de posibles fuentes de peligros o accidentes. Contiene además una sección sobre diseño y fabricación, que dedica un breve análisis al diseño de sistemas robotizados, teniendo en cuenta aspectos mecánicos, ergonómicos y de control. La mayoría de las indicaciones que se proporcionan son de carácter general.

5.3.2. Impacto socio-económico

El uso de robots es aceptado cada vez más y en los últimos años el mercado de los robots, en concreto los robots de servicio que es el campo al que pertenece la robótica asistencial, está en pleno auge.

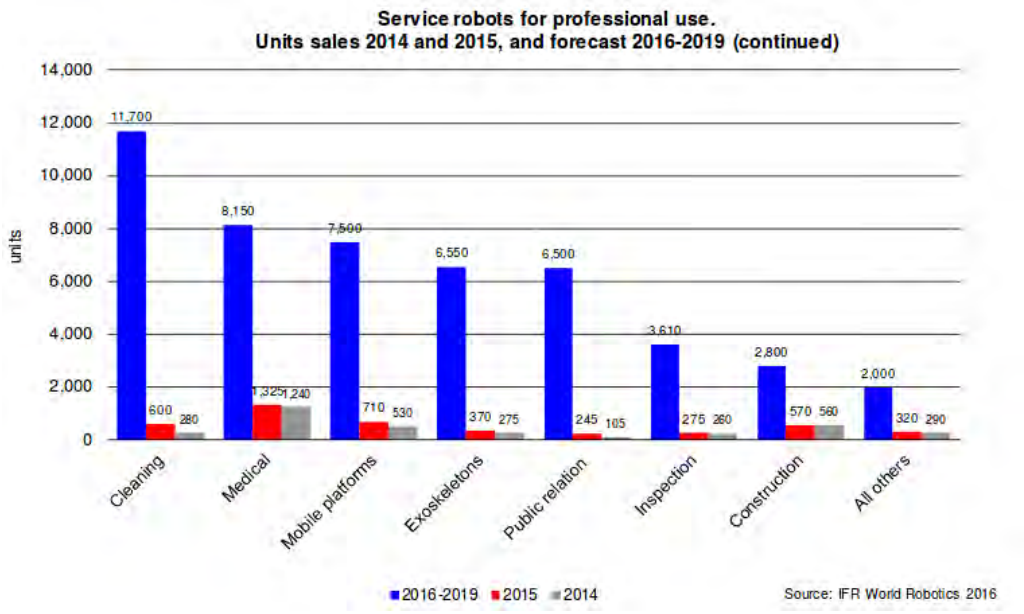


Figura 5.1: Robots de servicio para uso profesional

Según las previsiones del IFR World Robotics 2016, figura 5.1, en los próximos años el sector va aumentar exponencialmente y esto se debe a varios factores. Primero, la robótica se está desarrollando cada vez más deprisa por lo que los robots son cada vez más complejos y avanzados e incluso más baratos que antes.

Por otro lado, nuestra sociedad está cada vez más acostumbrada al uso de internet y las nuevas tecnologías por lo que ha disminuido el rechazo que pudiera surgir debido al aumento del número de robots. Además también hay que tener en cuenta nuestra sociedad es cada vez más y más mayor.

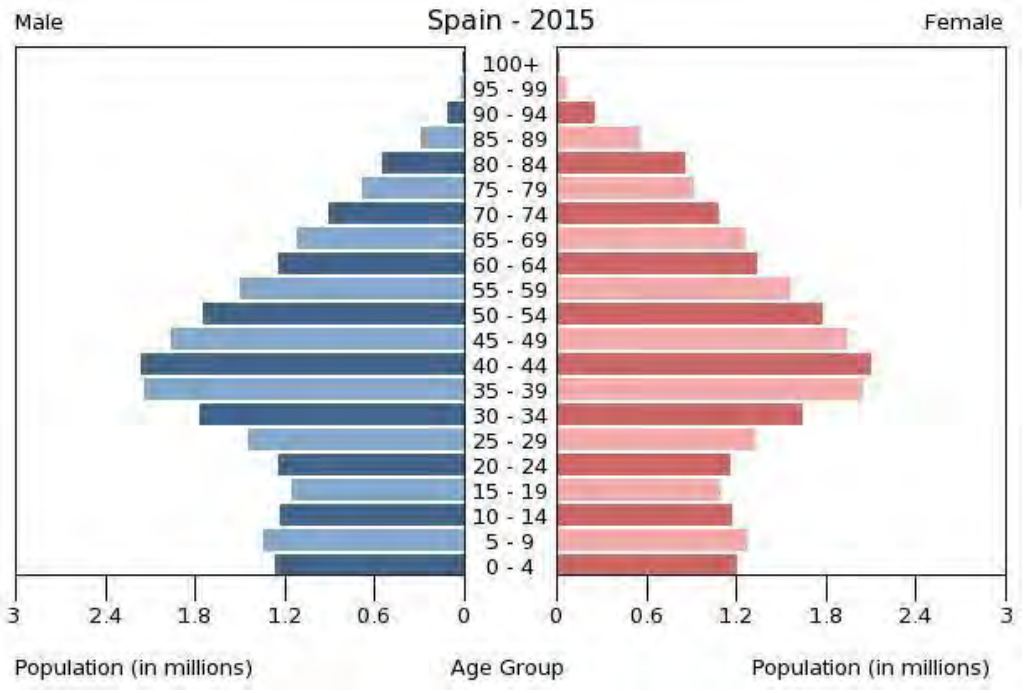


Figura 5.2: Pirámide de población en España (2015)

Por último, en la figura 5.2 se encuentra la pirámide de población de España en 2015 e indica claramente el envejecimiento de la sociedad. Y esto influye enormemente en la robótica asistencial porque a medida que el ser humano vaya envejeciendo va a requerir de ayuda y apoyo para su día a día porque ya no sería capaz de realizar algunas tareas sin ayuda y ese es el objetivo de este área de la robótica.

5.4. Presupuesto del proyecto

En cuanto a los gastos se han dividido en varias áreas: Costes de Mano de Obra, Costes de Amortización, Costes de Licencias, Costes de Material y los Costes Indirectos.

En los **costes de mano de obra**, tabla 5.1, se ha realizado una estimación de las horas empleadas tanto del alumno como de la persona encargada de supervisar los resultados y del precio que tendrían. Por otra parte en los **costes de amortización**, tabla 5.2, abarca el coste del uso tanto de AMOR como de Leap Motion pasando por el propio ordenador que se utiliza para el diseño del trabajo. A continuación el **coste de licencias**, tabla 5.3, es nulo porque se ha empleado software libre (ROS, Ubuntu, Latex) y el **coste de los materiales**, tabla 5.4, también porque sólo se

ha empleado AMOR, Leap Motion y un ordenador que se encuentran en la parte de amortización. Por último , en **costes indirectos**, tabla 5.5, se encuentra el internet y la electricidad utilizada.

COSTES DE MANO DE OBRA			
Descripción	**Tiempo** (h)	**Precio/hora (€/h)**	**Coste (€)**
Alumno de ingeniería	400	10	4000
Personal supervisión resultados	2	25	50
Total			4050

Tabla 5.1: Costes de la mano de obra

COSTES DE AMORTIZACIÓN				
Descripción	**Precio (€)**	**Vida** ($años$)	**Horas utilizadas** (h)	**Coste (€)**
Ordenador Portátil	700	10	350	2.8
robot AMOR	25.000	10	100	28.54
Leap Motion	70	5	150	0.24
Total				31.58

Tabla 5.2: Costes de amortización

COSTES DE LICENCIAS	
Descripción	**Coste (€)**
-	0
Total	0

Tabla 5.3: Costes de licencias

COSTES FIJOS DE MATERIAL			
Descripción	**Unidades**	**Precio/unidad (€)**	**Coste (€)**
-	0	-	0
Total			0

Tabla 5.4: Costes fijos de material

COSTES INDIRECTOS			
Descripción	**Precio/mes (€/mes)**	**Uso (h)**	**Coste (€)**
Internet	24	250	8.33
Corriente eléctrica	60	350	29.16
Total			37,5

Tabla 5.5: Costes indirectos

COSTES TOTALES	
Descripción	**Coste (€)**
Costes de mano de obra	4050
Costes de licencias	-
Costes fijos de material	-
Costes de amortización	31.58
Costes indirectos	37.5
Total	4119.08

Tabla 5.6: Costes totales del proyecto

Bibliografía

[1] "Estadísticas sanitarias mundiales," tech. rep., Organización mundial de la salud, 2014.

[2] R. A. Española, *Definición de Robótica*. http://dle.rae.es/?id=WYTm4uf, [Ultimo acceso: Septiembre 2017].

[3] I. F. of Robotics, *History of robots*. https://ifr.org/robot-history, [Ultimo acceso: Septiembre 2017].

[4] . Normas ISO/TR 11065:1992.

[5] Y. R. et al., "Use of a therapeutic, socially assistive pet robot (paro) in improving mood and stimulating social interaction and communication for people with dementia: Study protocol for a randomized controlled trial," *JMIR Res Protoc*, 2015.

[6] S. H. H. y K M Goher, "Personal care robots for older adultuts: An overview," *Asian Social Science*, 2016.

[7] ROS.org, *What is ROS?* http://wiki.ros.org/ROS/Introduction, [Ultimo acceso: Septiembre 2017].

[8] A. Ademovic, "An introduction to robot operating system: The ultimate robot application framework."

[9] ROS.org, *Concepts*. http://wiki.ros.org/ROS/Concepts, [Ultimo acceso: Septiembre 2017].

[10] A. Davis, *Getting Started with the Leap Motion SDK*. http://blog.leapmotion.com/getting-started-leap-motion-sdk/, [Ultimo acceso: Septiembre 2017].

[11] M. Shah and F. Lier, *ROS Leap Motion*. https://github.com/qqfly/leap_motion, [Ultimo acceso: Septiembre 2017].

[12] Exact Dynamics BV, *AMOR Service Manipulator: Instructions for Use*, May 2012.

[13] A.-S. J. y Montañés J.L., "Development of a robotic arm and implementation of a control strategy for gesture recognition through leap motion device," Master's thesis, Department of Electrical Engineering. School of Engineering and Architecture, EINA, 2016.

[14] F. D. Giulio Marin and P. Zanuttigh, "Hand gesture recognition with leap motion and kinect devices," in *2014 IEEE International Conference on Image Processing (ICIP)*, pp. 1565–1569, 2014.

[15] T. Wei Lu Zheng and J. Chu, "Dynamic hand gesture recognition with leap motion controller," *IEEE Signal Processing Letters*, vol. 23, no. 9, pp. 1188–1192, 2016.

[16] L. Shao, "Hand movement and gesture recognition using leap motion controller," course report, Stanford University, 2016.

[17] D. B. et at., "Intuitive and adaptive robotic arm manipulation using the leap motion controller," in *ISR ROBOTIK*, 2014.

[18] C. G. et al., "Home environment interaction via service robots and the leap motion controller," in *ISARC*, pp. 795–803, 2014.

[19] C. de Asusntos Jurídicos, "Proyecto de informe con recomendaciones destinadas a la comisión europea sobre normas de derecho civil sobre robótica," tech. rep., Parlamento Europeo, 31/5/2016.

[20] C. B. A. Barrientos, L. F.Peñin and R. Aracil, *Fundamentos de Robótica*. 2º edición, 2007.

[21] J. M. O'Kane, *A Gentle Introduction to ROS*. University of South Carolina, 2016.

Apéndice

Guía para la utilización del sistema

Para instalar todo hay que seguir los pasos descritos en:

https : *//github.com/gendibal*784*/AMORcontrol.git*

Una vez instalado, compilado y activado todo hay 5 programas: El publisher de Leap Motion, el reconocimiento gestual, el cambio de modo, el programa principal de control y programa de control de AMOR

1. Con *roscore* se habilitan los programas de ROS. En otro terminal, se lanza *yarpserver –write –ros* para conectar YARP con ROS y habilitar el programa de AMOR. Con *sudo leapd* se inicia Leap Motion.

2. A continuación con *rosrun leap_motion sender.py* se lanza el publisher de Leap Motion en un terminal.

3. Con *rosrun gesture gestos* se ejecuta el reconocimiento gestual.

4. Con *./final* se lanza el programa de control de AMOR

5. *rosrun gesture cambio* habilita el cambio de modo

6. Por último *rosrun amorpos control* lanza el programa principal de control.

nombre del dato			
	director	x	Tipo float64
		y	Tipo float64
		z	Tipo float64
	normal	x	Tipo float64
		y	Tipo float64
		z	Tipo float64
	palmpos	x	Tipo float64
		y	Tipo float64
		z	Tipo float64
	ypr	x	Tipo float64
		y	Tipo float64
		z	Tipo float64
	dedo_metacarpal[1]	x	Tipo float64
		y	Tipo float64
		z	Tipo float64
	dedo_proximal[1]	x	Tipo float64
		y	Tipo float64
		z	Tipo float64
	dedo_intermediate[1]	x	Tipo float64
		y	Tipo float64
		z	Tipo float64
	dedo_distal[1]	x	Tipo float64
		y	Tipo float64
		z	Tipo float64
	dedo_tip[1]	x	Tipo float64
		y	Tipo float64
		z	Tipo float64

[1] dedo equivale a thumb, index, middle, ring y pinky, es decir, hay 5 vectores como esos por cada dedo

Tabla 5.7: Leapros

nombre del dato			
	linear	x	Tipo float32
		y	Tipo float64
		z	Tipo float64
	angular	x	Tipo float64
		y	Tipo float64
		z	Tipo float64

Tabla 5.8: Geometry_msgs/Twist